



TIBCO iProcess® Workspace (Browser)

Configuration and Customization

*Version 11.9.0
August 2022*



Contents

Contents	2
Introduction	12
Overview	12
iProcess Client	12
iProcess Client Launch Fragment	13
Custom Application Built with Components	15
Configuration Files	15
User Access	17
User Access Profiles	17
Hierarchy	20
Using a Single Profile for Multiple User Types	20
Access Profile 'name' Attributes	20
Creating Custom User Access Profiles	36
Configuring the Client Application	38
Server Nodes	38
Action Processor URL	41
Session Monitor	43
Hide Case Data Tab Find Tool	44
Remember Login Information	45
Customizing the Browser Window Caption	46
Customizing the Work Item Caption	48
Specifying Browser Window Features	49
Form Type	50
Browser Feature Attributes	51
Releasing Resources on Logout	57

Redirecting Client to URL on Logout	57
Redirecting Client to URL on Browser Session Timeout	58
User Options	59
Limiting Number of Cases	69
Setting the Maximum Number of Case History Entries	70
Specifying Default Page Size for Work Item Lists	71
Specifying Default Types/Statuses to Display on Lists	71
Server-Side Atomic Locking of Work Items	73
Specifying Whether Case Counts Should be Obtained	75
Specifying Outstanding Work Item Step Types	76
WebDAV Root Setting	77
Add-ins	78
TIBCO Forms Caching	78
Show/Hide Personal Work Queues	79
Refresh Work Item List After Work Item is Opened Externally	80
Customizations	81
Font and Image Settings	81
Adding Custom Menu Items and Toolbar Buttons	82
Extending User Access Profiles to Control Custom Menus and Toolbar Buttons	85
Callout Interface	88
Sample Callout Handler	90
Helper Function	91
Configuration	91
Callout Methods	92
overrideFilterFields	93
overrideInitialFilter	95
overrideFilter	97
overrideInitialHistoryFilter	99
overrideHistoryFilter	100
overrideSortFields	101

overrideInitialSort	103
overrideSort	105
overrideSelectColumns	107
overrideInitialColumns	110
overrideColumns	112
modifyMatrixColumns	114
Browser File Cache Issues	117
Browser File Cache Settings	117
How Expiration Dates Are Used	119
Clearing the Local Browser Cache	119
Content Expiration Dates on IIS	119
Other Considerations and Recommendations	120
Creating New Application Directory for Updates	120
Dynamic Work Item Status Icons Based on Priority	121
Dynamic Row Colors on Work Item List	128
Configuring the Action Processor	132
Overview	132
Log Settings	133
XML Response Compression	133
Return Request Parameters	134
External Form URI	134
Obfuscating External Form URI Information	135
Server Factories	136
XML Validation	137
CSRF Validation	137
Action Processor Version	138
Application Server Settings	139
Session Timeout	139
Maximum POST Size	142
Character Encoding	142

Java Heap Size	143
Direct Login	144
Direct Login	144
Enabling Direct Login	146
On the URL	146
In an HTML Form Element Named 'DirectLogin'	147
In an HTML Script Element that Defines 'getDirectLoginArgs'	148
Single Authentication	149
Introduction	149
Java Single Authentication	150
Web Server Configuration	150
Authenticator Plug-in	151
iProcess Workspace (Browser) Configuration	151
Java Single Authentication Sample	152
Setting Up the Java Single Authentication Sample	153
Launching the Java Single Authentication Sample	154
.NET Single Authentication	156
Web Server Configuration	156
Authenticator Plug-in	157
iProcess Workspace (Browser) Configuration	158
.NET Single Authentication Sample	159
Setting Up the .NET Single Authentication Sample	160
Launching the .NET Single Authentication Sample	161
Logging	163
Introduction	163
Application Log	163
Application Monitor	165
Localization	168
Localizing the iProcess Workspace (Browser)	168

Create a New Localized Language Resource File	169
Configure the New Localized Language in the iProcess Workspace (Browser)	171
Modify or Create a TIBCO® General Interface System Locale File	171
Translate User Access Profiles Descriptions	175
Set the New Default Language for the iProcess Workspace (Browser)	175
Create a New Folder to Hold Localized Help Files	176
IPC Tools Methods	177
Introduction	177
Method Summary	178
ipcStartCase	180
ipcShowCase	181
ipcCloseCases	182
ipcPurgeCases	183
ipcSuspendCases	184
ipcActivateCases	185
ipcShowGraphicalCaseHistory	186
ipcAddCaseHistoryEntry	188
ipcShowCasePrediction	189
ipcTriggerEvent	191
ipcProcessJump	193
ipcOpenWorkItem	195
ipcOpenWorkItemEx	196
ipcUnlockWorkItem	198
ipcForwardWorkItem	199
ipcReleaseWorkItem	201
ipcConfigureSupervisors	202
ipcConfigureParticipation	204
ipcConfigureRedirection	205
ipcShowWorkQLoadingChart	207
ipcGetStartProcs	209
ipcGetAuditProcs	209

ipcShowProcLoadingChart	210
ipcShowProcVersion	212
ipcShowServerInfo	213
ipcShowOptions	215
ipcWorkItemTag2CaseTag	217
ipcWorkItemTag2WorkQTag	218
ipcGetUserAttributes	218
ipcGetGroupAttributes	219
ipcManageMER	219
IPC Tools Methods Sample	220
Forms	225
Introduction to Forms	225
External Forms / GI Forms	226
TIBCO® General Interface Forms Interface	229
Overview	229
Base Class	231
Sample Implementation	231
Implementation	233
Interface Properties and Methods	237
Base Class Properties	237
Base Class Methods	240
buildCDFArrays	241
closeForm	242
confirmUserMessage	243
createFieldDefsRequest	244
createKeepRequest	248
createLockRequest	252
createReleaseRequest	257
doCancel	261
doClose	262

doKeep	263
doRelease	264
getWindowContext	265
init	266
lockWorkItem	267
onBeforeUnload	270
postLoadInit	270
readFieldDefs	273
readFormFields	275
readStepMarkings	278
showUserMessage	280
socketRequest	281
transformData	282
FieldData Class	283
FieldData Class Functions	284
Requesting Values For Items in an Array Field	286
Date Conversions	287
Code Example	290
Date Conversion Methods	291
jsDateToLocal	291
jsDateToXml	292
localToJSDate	293
localToXml	294
xmlToJSDate	295
xmlToLocal	295
Date Format Localization Methods	296
getAbbrMonthNames	297
setAbbrMonthNames	297
getAmPm	298
setAmPm	299
getFullMonthNames	300
setFullMonthNames	301

Accessing User Options When Using GI Forms	301
ASP Forms	304
ASP Form Example	304
Setting Up the ASP Form Project in IIS	304
Configuring iProcess Workspace (Browser) to Use the ASP Form	306
ASP Form Interface	310
JSP Forms	313
JSP Form Example	313
Configure iProcess Workspace (Browser) to Use the JSP Form	314
JSP Form Interface	317
Customizing iProcess Modeler Forms	320
Overview	320
Embedding HTML	321
Word Wrap in the Editor	322
Pre-Formatting of the Form	322
Disabling Pre-Formatting	322
Including Scripts	322
Nesting of HTML Tags with Conditional Statements	323
Functions Available for Embedded Scripting	323
Altering the Style of Various Controls	324
Embedded Customization Examples	326
File Caching	328
Setting up a Test Environment	329
Structure of the Complete iProcess Modeler Form Page	330
Functions Available for File-Cached Scripting	331
HTML for Marking Controls	334
File-Cached Customization Example	338
Common Issues for Embedded and File-Cached Customizations	343
Displaying Forms Outside of the iProcess Workspace (Browser)	345

The LinkForm Example	345
Using Forms with User Tasks	348
Overview	348
Creating a New Form	348
Creating a New Form for an Existing User Task	349
Creating a New Form Manually from the Project Explorer	351
Linking a Free-Standing Form to a User Task	353
Switching back to the Default Form	355
Updating Forms with Synchronization Wizard	357
Using Data Fields and Parameters with Process User Tasks	365
The Mode Property of User Task Parameters	365
Using Data Fields and Parameters	366
Data Types for Data Fields and Process Parameters	366
Adding iProcess System Fields to a Form	369
Layout Properties for Panes	370
Localizing a Form	371
Styling Forms Using Cascading Style Sheets	372
Setting CSS Classes	372
Using an External CSS Resource	372
Best Practices	372
Examples	373
Cascading Style Sheets - Advanced Samples	374
Import the Forms Advanced Samples	374
Using CSS to Customize the Rendering of a Form Control	374
Changing the Background Color of a Control based on its Value	377
TIBCO iProcess System Fields	379
Validation Errors	381
General Interface Errors	381
Using the GI Preview Mode on Linux	382
Upgrading to the Current Version of Mozilla Gecko on Linux	382
Deploying Process Packages, Forms, and Dependent Resources	383

Deploy a Process Package to the iProcess Server	383
Deploy a Form to the iProcess Workspace Server	384
Deploy Using the Deployment Wizard	386
Supported Forms Formats	386
Unsupported Forms Features	387
Managing Monitor Event Request (MER) Messages	388
Introduction	388
Manage Monitor Event Request Tool	390
Create New Monitor Event Request	390
Edit Monitor Event Request on the Server	394
Adding a New MonitorDetail to an Existing MER	396
Editing an Existing MonitorDetail in the MER	399
Import Monitor Event Request XML	401
Deprecated Callout Interface	403
Callout Interface	403
Configuration	408
Callout Method Signatures	411
TIBCO Documentation and Support Services	426
Legal and Third-Party Notices	428

Introduction

This section provides an introduction to configuring and customizing TIBCO iProcess Workspace (Browser).

Overview

All of the information in this guide can be used to configure and customize either of the following applications:

- iProcess Client
- Custom application built with components

iProcess Client

The iProcess Client is an application that is provided with TIBCO iProcess Workspace (Browser) that allows you to perform functions such as start cases of iProcess procedures, display case history, view work items in their work queue, etc.

Note, however, the iProcess Client was *not* built using the TIBCO iProcess Workspace (Browser) components. Therefore, it cannot be opened in TIBCO® General Interface Builder, like a custom application built with components, nor can you use component-specific items like WCC methods (which are described in the *TIBCO iProcess Workspace (Browser) Components Reference Guide*), to customize the iProcess Client.

The iProcess Client can be configured and customized only to the extent of what is described in this document.

Details about the functions available in this application can be found in the *TIBCO iProcess Workspace (Browser) User's Guide*.

iProcess Client Launch Fragment

When the iProcess Client is installed, a launch fragment is provided that is used to launch the iProcess Client application. This launch fragment is located as follows:

```
InstallDir\iProcessClient.html
```

where *InstallDir* is the directory specified during the installation of TIBCO iProcess Workspace (Browser).

After all installation and configuration tasks are completed, and the Web Application Server hosting the iProcess Client and Action Processor has been started, the iProcess Client can be launched by pointing a browser at the iProcess Client launch fragment:

```
http://Host:Port/ClientDir/iProcessClient.html
```

where:

- *Host* is the machine name on which the iProcess Client is being hosted.
This can also be the IP address (either an IPv4 or IPv6 address) of the host machine. Note, however, if it is an IPv6 address, it must be enclosed in square brackets when in a URL. For example:
`http://[2001:db8:85a3:8d3:1319:8a2e:370:7348]:port/restOfURL`
- *Port* is the port number used by the Web server to communicate with web applications.
- *ClientDir* is the directory (or virtual directory alias) in which you installed the iProcess Client files (which defaults to **TIBCOiPCInt**).

For example:

```
http://Roxie:8090/TIBCOiPCInt/iProcessClient.html
```

Cross-Domain Scripting

Cross-domain scripting is a security vulnerability of web applications. If you trigger cross-domain scripting, and your browser doesn't allow it, the web application will not run (in the case of an iProcess Workspace (Browser) application, it will state that it is unable to establish a connection to the Action Processor).

Some browsers are more strict about enforcing cross-domain scripting than others; and newer versions of browsers tend to be more strict than older versions. Some browsers

also provide methods to allow cross-domain scripting—see your browser’s documentation for more information.

Cross-domain scripting affects accessing iProcess Workspace (Browser) applications in the following ways:

- **URL used to launch the application** - To prevent cross-domain scripting, it is best practice to ensure that the domain portion of the URL that is entered into the address line of the browser *exactly* matches the domain portion of the Action Processor URL specified in the application’s `config.xml` file.

The domain consists of the "`http://Host:Port`" part of the URL.

The domain used to launch the application cannot differ in any way from the Action Processor’s specified domain, otherwise cross-domain scripting may be triggered (depending on your browser). That is, you cannot use "`http`" in one and "`https`" in the other; you cannot use a host name in one and an IP address in the other; one host name cannot be unqualified and the other qualified; you cannot use "`localhost`" in one and "`127.0.0.1`" in the other.

To determine if cross-domain scripting is being used, the browser simply compares the URL domains as strings.

- **Running the application from the local file system** - Because of the security risk of cross-domain scripting, some browsers will not allow you to run a web application (including the iProcess Client) from the local file system.

Note that you would typically only run the iProcess Client or a WCC application from the file system in a testing and development environment. In a production environment, it is expected that the application will be deployed to a Web server and run from there.

Launching the iProcess Client in an HTML Frame

To be able to launch the iProcess Client in an HTML frame (for example, an `iframe` in a portal), you must make some modifications to the launch fragment for the application.

Prior to launching the iProcess Client in a frame:

1. Open the launch fragment.
2. Locate the following: “NOTE: To allow display of this application under frames remove the following style and script elements”.
3. Remove (or comment out) the `<style>` and `<script>` elements immediately after the note. For example:

```

<!-- NOTE: To allow display of this application under frames remove the following style and script
elements
<style type="text/css">html{display:none}</style>
<script language="javascript">
  if (self == top) {
    // Not in frame so show client app
    document.documentElement.style.display='block';
  } else {
    // In a frame so try to show client app outside of a frame

```

4. Locate the following: “NOTE: To allow display of this application under frames remove the next script element”.
5. Remove (or comment out) the **<script>** element immediately after the note (do not, however, remove the second **<script>** element following the note). For example:

```

<!-- NOTE: To allow display of this application under frames remove the next script element
<script language="javascript">
  if (self != top) {
    // Still in a frame so clear body of app and close
    document.getElementsByTagName("body")[0].innerHTML = "Not allowed in frames.";
    window.open('close.html', '_self');
  }
</script> -->
<script type="text/javascript" src="JSX/js/JSC00.js"

```

6. Save and close the launch fragment.

Custom Application Built with Components

Custom applications that are built with TIBCO iProcess Workspace (Browser) components¹ can be configured and customized using any of the parameters described in this document. Plus, they can further enhanced using things like WCC methods and the JavaScript Interface, which are described in the *TIBCO iProcess Workspace (Browser) Components Reference Guide*.

Also see the *TIBCO iProcess Workspace (Browser) Components Concepts Guide* for information about how to create a custom application using components.

Configuration Files

There are two primary configuration files provided to configure the client application:

- **userAccessProfiles.xml** - This is used to specify which users have access to the functions available in the client application.

For information about configuring user access profiles, see [User Access](#).

¹Also commonly called “WCC” components.

- **config.xml** - This is used to configure many other aspects of the client application, such as which servers the user can connect to, default settings in the application, etc.

For information about configuring the client application, see [Configuring the Client Application](#).

The location of these configuration files depends on whether you are using the client application provided with the iProcess Workspace (Browser), or a custom application developed with the iProcess Workspace (Browser) components.

- If you are using the client application, these configuration files are located as follows:

```
ClientInstallDir\JSXAPPS\ipc\
```

where *ClientInstallDir* is the directory in which the client application is installed.

- If you are using a custom application developed with the iProcess Workspace (Browser) components, these configuration files are located as follows:

```
WorkspaceDir\JSXAPPS\ProjectName\
```

where *WorkspaceDir* is the directory that was designated as your workspace when TIBCO® General Interface (GI) Builder was initially started, and *ProjectName* is the name that was given to your GI Builder project when your application was developed with the components.

The references to these configuration files in this section assume you are configuring the client application provided with the iProcess Workspace (Browser).

If you are configuring a custom application created with the iProcess Workspace (Browser) components, substitute the path shown with the appropriate path.

User Access

This section describes setting up user access in the TIBCO iProcess Workspace (Browser).

User Access Profiles

User access profiles provide the ability to specify which application functionality is available to various types of users of the client application. They do this by specifying which user interface components (i.e., icons, buttons, and menu selections) are made available to the logged-in user.



Note

User access profiles only define which user interface components are made available to the logged-in user — the ability to actually execute the functionality is determined by the level of security defined on the iProcess Objects Server. For instance, the user's access profile may grant access to the tool/menu selection for closing cases, however, if the user does not have system administrative privileges on the iProcess Objects Server, any attempt to close a case will be rejected.

The user access profiles are defined using the **UserAccessProfiles** record in the *ClientInstallDir\JSXAPPS\ipc\userAccessProfiles.xml* file.

Each profile represents a type of application user and defines the user interface components available to users of that type. The following shows a collapsed view of the default user access profiles included in the iProcess Workspace (Browser):

```

<record jsxid="UserAccessProfiles" type="ipc">
  <Profiles serverUserAttr="MENUName">
    <Profile type="Default" description="Access Level: Default"> ...
    <Profile type="Admin" description="Access Level: Admin"> ...
    <Profile type="User" description="Access Level: General"> ...
    <Profile type="ProDef" description="Access Level: Definer"> ...
    <Profile type="Manager" description="Access Level: Manager"> ...
    <Profile type="PreLogin" description="Access Level: PreLogin"> ...
  </Profiles>
</record>

```

Each user's profile type is stored in the MENUName user attribute, the name of which is specified by the **serverUserAttr** attribute (for information about the MENUName user attribute, see the *TIBCO iProcess Server Objects Programmer's Guide* or on-line help system). By default, the MENUName attribute is used because it is an inherent attribute of all iProcess users and requires no customization when the iProcess Workspace (Browser) is installed.

The "Default" profile type is assigned to application users that do not have their iProcess attribute set to one of the defined profile types. In this example, if a user logs in to the iProcess Workspace (Browser), and the value of their MENUName iProcess attribute is empty, or set to a value other than "Admin", "User", "ProDef", or "Manager", the access defined for the "Default" profile type is assigned. If the MENUName value is invalid, and the "Default" profile type has not been defined in userAccessProfiles.xml, access is automatically limited to viewing only the list of procedures.

There is also a special "PreLogin" user type specified in the userAccessProfiles.xml file that represents all users before they login, i.e., before the application knows their user name/type. The access profile for the "PreLogin" user type only contains the elements needed to specify how much error information will be shown to the user prior to logging in.

If you need additional user profile types, you must create a new user attribute and assign profile types to that user attribute (rather than assigning new types to the MENUName attribute). For information on how to create custom profiles by defining a new user attribute, see [Creating Custom User Access Profiles](#).

Each user access profile (i.e., each **<Profile/>** element) specified in the userAccessProfiles.xml file contains the following attributes:

- The **type** attribute of each profile represents the user type and corresponds to the value that is stored in the iProcess attribute of the user. For example:

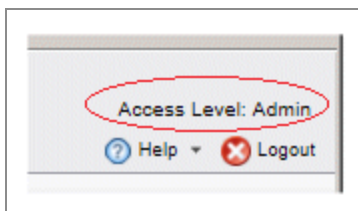
```
<Profile type="Admin" description="Access Level: Admin">
```

Initially, profiles are defined for each of the possible MENU_NAME values: “Admin”, “User”, “ProDef” and “Manager” (as well as a “Default” and “PreLogin” type, which are described above).

- The **description** attribute defines a text string that is displayed in the header area of the iProcess Workspace (Browser) interface, and provides an indication of the access level of the logged-in user. For example:

```
<Profile type="Admin" description="Access Level: Admin">
```

This example would cause the following to be displayed when a user with a MENU_NAME of “Admin” is logged in:



Each **<Profile/>** element contains subordinate **<property/>** elements, each of which represents a specific function in the iProcess Workspace (Browser). The **<property/>** elements contain the following attributes:

- The **name** attribute identifies the function for which you can provide or deny access using the **state** attribute (see the next bullet item).

```
<property name="Procedure" state="1">
```

For a complete list of the allowable **name** attributes (functions), see the table in the [Access Profile ‘name’ Attributes](#) section on [Access Profile ‘name’ Attributes](#).

- The **state** attribute specifies whether or not the associated user type has access to the functionality identified by the **name** attribute (see the bullet item above), where “1” means allow access and “0” means deny access.

```
<property name="Procedure" state="1">
```

If access to a function is not allowed, the applicable buttons and/or menu selections are not displayed.

Note that if a **<property/>** element for a particular function is not present in the `userAccessProfiles.xml` file, access to that function is not allowed by default.

Hierarchy

The hierarchy of the **<property/>** elements is significant, i.e., if a child **<property/>** element gives the user type access to a button/menu selection, the child's parent **<property/>** element must also be enabled (by setting its **state** attribute to "1"). For example, see the following excerpt from the `userAccessProfiles.xml` file. If the user type is given access to open work items (name="Open"), it must also be given access to view the work items (name="WorkItem").

```
<property name="WorkItem" state="1">
  <property name="AutoRefresh" state="1"/>
  <property name="Forward" state="1">
    <property name="ForwardAnyQueue" state="1"/>
  </property>
  <property name="Open" state="1"/>
  <property name="OpenFirst" state="1"/>
</property>
```

Likewise, if you give the user type access to "ForwardAnyQueue", you must also give access to "Forward", since "ForwardAnyQueue" is a child of "Forward".

Using a Single Profile for Multiple User Types

The same access profile can be used for multiple user types by including the optional **<Type/>** element. The example shown below for the **Admin** user illustrates an example of this — the **Admin2** user uses the same profile.

```
<!Profile types="Admin" description="Access Level: Admin">
  <Type name="Admin2" description="Access Level: Admin2"/>
  <property name="Procedure" state="1">
    <property name="Versions" state="1"/>
  </property>
  <property name="ManageMER" state="1"/>
  <property name="LoadingChart" state="1"/>
</Profile>
```

Access Profile 'name' Attributes

This section provides descriptions of each function for which you can control access using the user access profiles.

The table below provides a list of all of the **name** attribute values for the **<property>** element of an access profile, and the meaning of each.

The **'name' Attribute Values** column shows the hierarchy of **name** attributes within the profile.

'name' Attribute Values	Description
Procedure ¹	Provides access to the procedure list.
Procedure Versions	Provides access to the Procedure Versions tool on the procedure list.
Procedure ManageMER	Provides access to the Manage Monitor Event Request tool on the procedure list.
Procedure LoadingChart	Provides access to the Procedure Loading Chart tool on the procedure list.
Procedure CaseStart	Provides access to the Start New Case tool on the procedure list.
Procedure Status	Provides access to the following selections on the procedure list View menu: Released Procedures, Unreleased Procedures, Model Procedures, and Withdrawn Procedures . This allows you to control whether or not the user can choose which statuses of procedures to display.
Procedure Type	Provides access to the following selections on the procedure list View menu: Main Procedures, Sub-Procedures, and Main and Sub-Procedures . This allows you to control whether or not the user can choose which types of procedures to display.
Procedure Case	Provides access to the case list.

¹If a user that has neither procedure view access (name="Procedure") nor work queue view access (name="WorkQueue") logs into the iProcess Workspace (Browser), a screen is displayed containing the message: "Access to Procedure and Work Queue data is denied - please contact your system administrator". The only components available on this screen are the Logout and Help tools.

'name' Attribute Values	Description
Procedure Case Activate	Provides access to the Activate Case(s) tool on the case list, and the Activate Case tool on the Summary tab when the case is opened from the case list.
Procedure Case Close	Provides access to the Close Case(s) tool ¹ on the case list, and the Close Case tool on the Summary tab when the case is opened from the case list.
Procedure Case Jump	Provides access to the Process Jump tool on the case list and on the Summary tab when the case is opened from the case list.
Procedure Case Jump DataRead	Provides read-only access to Case Data dialog available through the Process Jump dialog. If DataUpdate access is also enabled, it overrides this element, giving the user update access to case data. If both this and DataUpdate access are disabled, the Data button is not displayed on the Process Jump dialog.
Procedure Case Jump DataUpdate	Provides update access to Case Data dialog available through the Process Jump dialog. (This overrides DataRead if it is also enabled.) If both this and DataRead access are disabled, the Data button is not displayed on the Process Jump dialog.
Procedure Case	Provides access to the Select Columns selection on the View menu for the outstanding items list on the Process Jump

¹The case close (name="Close"), case purge (name="Purge"), and work queue supervisors (name="Supervisors") functions all require system administrative authority. Without system administrative authority, a user cannot perform these functions even if their access profile provides access to the tools/buttons for these functions. If a user's access profile causes the buttons/selections to appear, but they don't have administrative authority, the buttons/selections are grayed out.

'name' Attribute Values	Description
SelectColumns	dialog.
Procedure Case Suspend	Provides access to the Suspend Case(s) tool on the case list, and the Suspend Case tool on the Summary tab when the case is opened from the case list.
Procedure Case Trigger	Provides access to the Trigger Events tool on the case list and on the Summary tab when the case is opened from the case list.
Procedure Case Trigger DataRead	Provides read-only access to Case Data dialog available through the Events dialog. If DataUpdate access is also enabled, it overrides this element, giving the user update access to case data. If both this and DataUpdate access are disabled, the Data button is not displayed on the Events dialog.
Procedure Case Trigger DataUpdate	Provides update access to Case Data dialog available through the Events dialog. (This overrides DataRead if it is also enabled.) If both this and DataRead access are disabled, the Data button is not displayed on the Events dialog.
Procedure Case Trigger Resurrect	Provides access to the Trigger Events tool on the case list when a closed case is selected, and on the Summary tab when a closed case has been opened from the case list.
Procedure Case	Provides access to the Recalculate Deadlines radio buttons on the Events dialog. ¹

¹If you are using an older iProcess Objects Server that does not provide recalculate deadline functionality, you may want to set state="0" for **RecalculateDeadlines** so that the **Recalculate Deadline** radio buttons are not displayed on the **Events** dialog.

'name' Attribute Values	Description
RecalculateDeadlines	
Procedure Case Purge	Provides access to the Purge Case(s) tool ² on the case list, and the Purge Case tool on the Summary tab when the case is opened from the case list.
Procedure Case Open	Provides access to the Open Case(s) tool on the case list.
Procedure Case Open Summary	Provides access to the case Summary tab when a case is opened from the case list.
Procedure Case Open History	Provides access to the case History tab when a case is opened from the case list.
Procedure Case Open History AddHistoryEntry	Provides access to the Add Entry tool on the History tab when the case is opened from the case list.
Procedure Case Open History Predict	Provides access to the Predict Case tool on the History tab when the case is opened from the case list.
Procedure Case Open History GraphicalHistory	Provides access to the Graphical History tool on the History tab when the case is opened from the case list.

'name' Attribute Values	Description
Procedure Case Open History FilterHistory	Provides access to the Filter History tool on the History tab when the case is opened from the case list.
Procedure Case Open Outstanding	Provides access to the case Outstanding tab when a case is opened from the case list.
Procedure Case Open Outstanding SelectColumns	Provides access to the Select Columns selection on the View menu on the case Outstanding tab.
Procedure Case Open DataRead	Provides read-only access to case data on the Data tab when a case is opened from the case list. If DataUpdate access is also enabled, it overrides this element, giving the user update access to data. If both this and DataUpdate access are disabled, the case Data tab is hidden.
Procedure Case Open DataUpdate	Provides update access to the case Data tab (this overrides DataRead if it is also enabled) when a case is opened from the case list. If both this and DataRead are disabled, the case Data tab is hidden.
Procedure Case SelectColumns	Provides access to the Select Columns selection on the View menu on the case list.
Procedure	Controls whether or not the user can click on the column

'name' Attribute Values	Description
SortableColumns	header to sort the case list.
Procedure Case Filter	Provides access to the Filter tool on the case list.
Procedure Case Sort	Provides access to the Sort tool on the case list.
Procedure Case Preview	Provides access to the Preview button and the Preview selection on the case list View menu.
Procedure Case Preview CasePreviewOn	Provides access to the Preview On - Open Details in Preview Pane selection from the Preview menu on the case list.
Procedure Case Preview CasePreviewFloat	Provides access to the Preview On - Float Details selection from the Preview menu on the case list.
Procedure Case Preview CasePreviewOff	Provides access to the Preview Off selection from the Preview menu on the case list.
Procedure SelectColumns	Provides access to the Select Columns selection on the View menu on the procedure list.
WorkQueue ¹	Provides access to the work queue list.
WorkQueue LoadingChart	Provides access to the Work Queue Loading Chart tool on the work queue list.

'name' Attribute Values	Description
WorkQueue Participation	Provides access to the Manage Work Queue Participation tool on the work queue list.
WorkQueue Redirection	Provides access to the Manage Work Queue Redirection tool on the work queue list.
WorkQueue Supervisors	Provides access to the Manage Work Queue Supervisors tool2 on the work queue list.
WorkQueue Status	Provides access to the following selections on the work queue list View menu: Released Work Queues, Test Work Queues, Released and Test Work Queues . This allows you to control whether or not the user can choose which statuses of work queues to display.
WorkQueue Type	Provides access to the following selections on the work queue list View menu: User Work Queues, Group Work Queues, and User and Group Work Queues . This allows you to control whether or not the user can choose which types of work queues to display.
WorkQueue PersonalWorkQueueOption	Specifies whether or not the user can show or hide their own personal work queue. If state ="1", a selection appears on the work queue list View menu that allows the user to either show or hide their personal work queue; if state = "0", the menu selection does not display. (Also see, Show/Hide Personal Work Queues .)
WorkQueue WorkItem	Provides access to the work item list.
WorkQueue WorkItem AutoRefresh	Provides access to the Auto-Refresh tool on the work item list.
WorkQueue	Provides access to the Forward Work Item(s) tool on the

'name' Attribute Values	Description
Forward	work item list.
WorkQueue WorkItem Forward ForwardAnyQueue	This is applicable only if the user has access to the Forward Work Item(s) tool. This causes the list of work queues on the Forward Selected Work Items dialog to contain all work queues on the system. If disabled, the list of work queues on the Forward Selected Work Items dialog will contain only the work queues of which the user is a member.
WorkQueue WorkItem Open	Provides access to the Open Selected Work Item(s) tool on the work item list.
WorkQueue WorkItem OpenFirst	Provides access to the Open First Work Item tool on the work item list.
WorkQueue WorkItem OpenNext	Provides access to the Open Next Work Item tool on the work item list.
WorkQueue WorkItem OpenAuto	Provides access to the Auto-Repeat Open Work Item tool on the work item list. Note that if access to both OpenFirst and OpenNext (see above) are prohibited, the Auto-Repeat Open Work Item tool is automatically disabled, as it requires OpenFirst and OpenNext.
WorkQueue WorkItem Release	Provides access to the Release Work Item(s) tool on the work item list. (If disabled, it does not prevent the user from releasing a work item via a form.)

'name' Attribute Values	Description
WorkQueue WorkItem Unlock	Provides access to the Unlock Work Item(s) tool on the work item list.
WorkQueue WorkItem SelectColumns	Provides access to the Select Columns selection on the View menu on the work item list.
WorkQueue WorkItem SortableColumns	Controls whether or not the user can click on the column header to sort the work item list.
WorkQueue WorkItem PageSize	Provides access to the Page Size selection on the work item list View menu.
WorkQueue WorkItem Preview	Provides access to the Preview button and the Preview selection on the work item list View menu.
WorkQueue WorkItem Preview WIPreviewOn	Provides access to the Preview On - Open Forms in Preview Pane selection from the Preview menu on the work item list.
WorkQueue WorkItem Preview WIPreviewFloat	Provides access to the Preview On - Float Forms selection from the Preview menu on the work item list.
WorkQueue WorkItem Preview WIPreviewOff	Provides access to the Preview Off selection from the Preview menu on the work item list.
WorkQueue WorkItem	Provides access to the Open Case tool on the work item list.

'name' Attribute Values	Description
OpenCase	
WorkQueue WorkItem OpenCase Summary	Provides access to the case Summary tab when a case is opened from the work item list.
WorkQueue WorkItem OpenCase Summary Activate	Provides access to the Activate Case tool on the Summary tab when a case is opened from the work item list.
WorkQueue WorkItem OpenCase Summary Close	Provides access to the Close Case tool2 on the Summary tab when a case is opened from the work item list.
WorkQueue WorkItem OpenCase Summary WiJump	Provides access to the Process Jump tool on the Summary tab when a case is opened from the work item list.
WorkQueue WorkItem OpenCase Summary WiJump DataRead	Provides read-only access to Case Data dialog available through the Process Jump dialog. If DataUpdate access is also enabled, it overrides this element, giving the user update access to case data. If both this and DataUpdate access are disabled, the Data button is not displayed on the Process Jump dialog.
WorkQueue WorkItem OpenCase Summary	Provides update access to Case Data dialog available through the Process Jump dialog. (This overrides DataRead if it is also enabled.)

'name' Attribute Values	Description
DataUpdate	If both this and DataRead access are disabled, the Data button is not displayed on the Process Jump dialog.
WorkQueue WorkItem OpenCase Summary WiJump SelectColumns	Provides access to the Select Columns selection on the View menu for the outstanding items list on the Process Jump dialog.
WorkQueue WorkItem OpenCase Summary Suspend	Provides access to the Suspend tool on the Summary tab when a case is opened from the work item list.
WorkQueue WorkItem OpenCase Summary WiTrigger	Provides access to the Trigger Events tool on the Summary tab when a case is opened from the work item list.
WorkQueue WorkItem OpenCase Summary WiTrigger DataRead	Provides read-only access to Case Data dialog available through the Events dialog. If DataUpdate access is also enabled, it overrides this element, giving the user update access to case data. If both this and DataUpdate access are disabled, the Data button is not displayed on the Events dialog.
WorkQueue WorkItem OpenCase Summary WiTrigger DataUpdate	Provides update access to Case Data dialog available through the Events dialog. (This overrides DataRead if it is also enabled.) If both this and DataRead access are disabled, the Data button is not displayed on the Events dialog.

'name' Attribute Values	Description
WorkQueue WorkItem OpenCase Summary WiTrigger Resurrect	Provides access to the Trigger Events tool on the Summary tab when you've opened a case from the work item list. Note that because closed cases cannot be seen in the work item list, access to this function is applicable only in the following situations: — After opening the case from the work item list, you close the case from the Case Details, then before closing the Case Details dialog, you decide to resurrect the closed case. — After opening the case from the work item list, another user closes the case while you still have the Case Details open. If you refresh the Case Details, the new "Closed" status will be displayed. At that point, you can resurrect the closed case.
WorkQueue WorkItem OpenCase Summary WiTrigger RecalculateDeadlines	Provides access to the Recalculate Deadlines radio buttons³ on the Events dialog when a case is opened from the work item list.
WorkQueue WorkItem OpenCase Summary Purge	Provides access to the Purge Case tool² on the Summary tab when a case is opened from the work item list.
WorkQueue WorkItem OpenCase WiHistory	Provides access to the case History tab when a case is opened from the work item list.
WorkQueue	Provides access to the Add Entry tool on the History tab

'name' Attribute Values	Description
OpenCase WiHistory AddHistoryEntry	when a case is opened from the work item list.
WorkQueue WorkItem OpenCase WiHistory Predict	Provides access to the Predict Case tool on the History tab when a case is opened from the work item list.
WorkQueue WorkItem OpenCase WiHistory GraphicalHistory	Provides access to the Graphical History tool on the History tab when a case is opened from the work item list.
WorkQueue WorkItem OpenCase WiHistory FilterHistory	Provides access to the Filter History tool on the History tab when a case is opened from the work item list.
WorkQueue WorkItem OpenCase WiOutstanding	Provides access to the case Outstanding tab when a case is opened from the work item list.
WorkQueue WorkItem OpenCase WiOutstanding SelectColumns	Provides access to the Select Columns selection on the View menu on the case Outstanding tab when a case is opened from the work item list.
WorkQueue WorkItem OpenCase DataRead	Provides read-only access to case data on the Data tab when a case is opened from the work item list. If DataUpdate access is also enabled, it overrides this

'name' Attribute Values	Description
	element, giving the user update access to data. If both this and DataUpdate access are disabled, the case Data tab is hidden.
WorkQueue WorkItem OpenCase DataUpdate	Provides update access to the case Data tab (this overrides DataRead if it is also enabled) when a case is opened from the work item list. If both this and DataRead are disabled, the case Data tab is hidden.
WorkQueue WorkItem Filter	Provides access to the Filter tool on the work item list.
WorkQueue WorkItem Sort	Provides access to the Sort tool on the work item list.
WorkQueue SelectColumns	Provides access to the Select Columns selection on the View menu on the work queue list.
SessionActivity	Provides access to the Session Activity button/icon.
SessionActivity ClearActivity	Provide access to the Clear button on the Session Activity dialog.
ServerInfo	Provides access to the Server Info button/icon.
Options	Provides access to the Options dialog.
Options Language	Provides access to the language selection field on the Options dialog.
Options InitialList	Provides access to the “Select which list initially displays at startup” selection on the Options dialog.

'name' Attribute Values	Description
Options WorkQueueReference	Provides access to the “Work queues may be referenced by name or description” selection on the Options dialog.
Options ProcedureReference	Provides access to the “Procedures may be referenced by name or description” selection on the Options dialog.
Options AutoRefresh	Provides access to the “Auto-refresh lists of work items” check box on the Options dialog.
Options WorkItemFilters	Provides access to the “Work Item Filters” radio buttons on the Options dialog.
Options CaseFilters	Provides access to the “Case Filters” radio buttons on the Options dialog.
Options CasePreview	Provides access to the “Case Preview Default” radio buttons on the Options dialog.
Options WorkItemPreview	Provides access to the “Work Item Preview Default” radio buttons on the Options dialog.
Options BrowserOrDialog	Provides access to the “When opening a floating work item form, open it in” radio buttons on the Options dialog.
Options SizeAndPosition	Provides access to the “Default position and size” fields on the Options dialog.
Options OutstandingItems	Provides access to the “Outstanding Items Options” radio buttons on the Options dialog.
Options SessionActivity	Provides access to the “Session Options” check boxes (which control the actions that are written to the Session Activity log) on the Options dialog.
Options ChangePassword	Provides access to the Change Password button on the Options dialog.

'name' Attribute Values	Description
Options SubCaseVersion	Provides access to the “Sub-Case” Version Options” radio buttons on the Options dialog.
ApplicationLog	Provides access to the iProcess Workspace (Browser) application log by pressing F12 . (For more information, see Application Log .) If disabled, the F12 function key has no function.
ChangePwdExpired	If enabled, this causes the Change Password dialog to be displayed when the user attempts to log in with an expired password. Requires a password change to log in.
ChangePwdOption	Provides access to the Change Password button on the Options dialog.
ShowErrorDetail	If enabled, details about error conditions are displayed to the user.
ShowErrorDetail ShowStackTrace	If enabled, a stack trace is shown when error information is displayed.

Creating Custom User Access Profiles

Custom user access profiles allow the client application interface to be tailored for various user categories that exist within an organization. The simplest form of customization is to modify the **state** attribute values for the existing MENUAME profile types that are defined at installation.

If the existing profile types are not sufficient to reflect the organization's user types, further customization may be performed. Rather than using the default MENUAME attribute to specify the profile of each user, a new iProcess user attribute must be defined and used to hold the name of customized user profile types. The customization process is as follows:

1. Define a new iProcess user attribute (e.g., ACCESS) on the server. The new iProcess attribute must be created by a user with administrative access using the TIBCO

iProcess User Manager (for information about the User Manager, see the *TIBCO iProcess Workspace (Windows) Manager's Guide*).

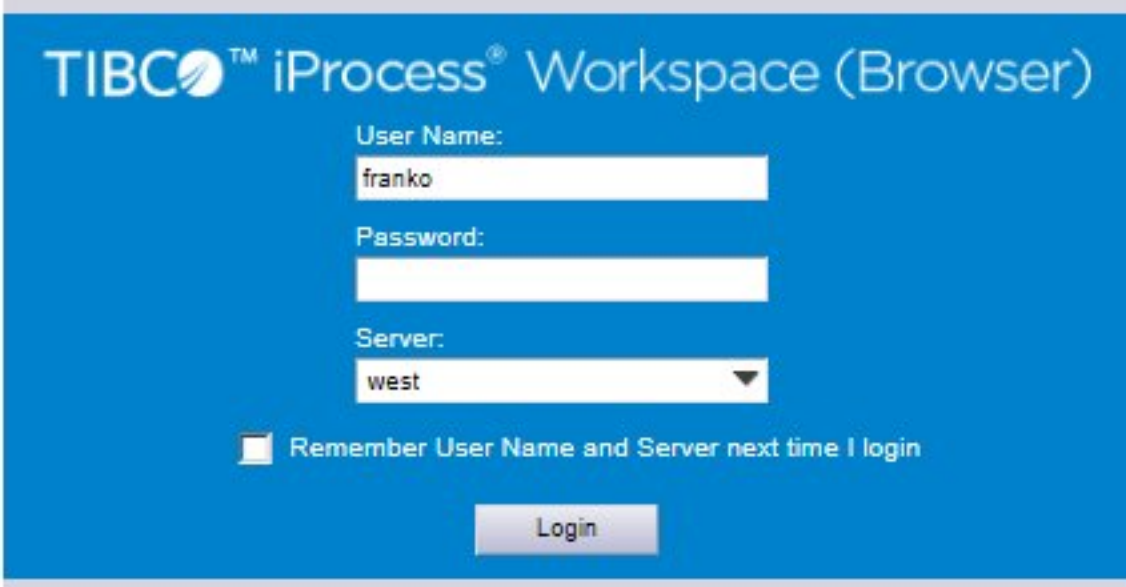
2. In the `userAccessProfiles.xml` file, change the value of the **serverUserAttr** attribute from “MENU_NAME” to the name of the new iProcess attribute created in step 1.
3. Modify the profile types and state values as desired. For information about how to do this, see [User Access Profiles](#).
4. For each iProcess user, set the value of their iProcess attribute to one of the profile types defined in `userAccessProfiles.xml`. This must be done by a user with administrative access using the TIBCO iProcess User Manager (for information about the User Manager, see the *TIBCO iProcess Workspace (Windows) Manager's Guide*).

Configuring the Client Application

This section describes configuration parameters that are available in the application's `config.xml` file.

Server Nodes

When the client application **Login** screen is displayed, the **Server** field drop-down list will contain a list of TIBCO iProcess Objects Servers that the user can log into:



The screenshot shows the login interface for TIBCO iProcess Workspace (Browser). It features a blue background with the TIBCO logo and the text 'TIBCO iProcess Workspace (Browser)'. There are three input fields: 'User Name:' with the value 'franko', 'Password:', and 'Server:' with a dropdown menu showing 'west'. Below these is a checkbox labeled 'Remember User Name and Server next time I login' and a 'Login' button.

This list of servers is controlled using the **ServerNodes** record in the client application's configuration file.

During the iProcess Workspace (Browser) installation process, you enter the information for one iProcess Objects Server. This section describes how to add additional entries to the **ServerNodes** record if you would like more than one server to appear in the **Server** field drop-down list.

To configure the server nodes for your installation:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).
2. Locate the **ServerNodes** record:

```
<record jsid="ServerNodes" type="ipc">
  <record displayName="Portland">
    <NodeId>
      <ComputerName>francine/ComputerName>
      <IPAddress>10.97.5.34/IpAddress>
      <TCPPort>52012/TCPPort>
      <Name>Corpe/Name>
      <Director>false/Directory>
    </NodeId>
    <UserPreferencePersistence persistOnServer="true"
      maxSize="32768"/>
  </record>
  <!--
  <record displayName="Server Two">
    <NodeId>
      <ComputerName>ComputerName/ComputerName>
      <IPAddress>255.255.255.255/IpAddress>
      <TCPPort>99999/TCPPort>
      <Name>Server2/Name>
      <Director>false/Directory>
      <UserPreferencePersistence persistOnServer="false"
        maxSize="32768"/>
    </NodeId>
  </record>
  <record displayName="Server Three">
```

The first record should reflect the entries that were entered during the installation. The elements in this record can be modified if the information is no longer correct (for instance, the TCP port has changed for that server).

Placeholders have been provided for two additional iProcess Objects Servers.

3. To configure additional servers, remove or move the appropriate comment delimiters from the “Server Two” or “Server Three” record, then enter the appropriate information in the following elements:
 - **displayNameName**: The name that you would like displayed in the iProcess Workspace (Browser) **Login to** field drop-down list. This is the name the user would select when choosing a server to log into.
 - **<ComputerName>**: The name of the machine on which the TIBCO iProcess Objects Server is installed.
 - **<IPAddress>**: The IP address of the machine on which the TIBCO iProcess Objects Server is installed. You can enter the name of the host machine in this field, as long as that name resolves to the IP address of the machine where the iProcess Objects Server is running. Note, however, that this name must be able to be resolved by the machine on which the Action Processor is running.

Note that an IPv6 IP address can be used if the TIBCO iProcess Objects Server uses IPv6 addressing (an example IPv6 address is: `he80::c531:b7a1:8922:ab12%13`).
 - **<TCPPort>**: The TCP port number used by the TIBCO iProcess Objects Server. (The TCP port used by the server is specified using the iProcess Objects Server

Configuration Utility in Windows systems (*SWDIR\bin\SWEntObjSvcCfg.exe*), or by editing the iProcess Objects Server configuration file in UNIX systems (*\$SWDIR/seo/data/swentobjsv.cfg*). For more information, see the *TIBCO iProcess Objects Server Administrator's Guide*.)

- **<Name>**: The name of the TIBCO iProcess Engine / iProcess Objects Server to which the user can log in. This is the “nodename” that is assigned to the iProcess Engine when it is installed.
- **<Director>**: Specifies whether or not the previous entries actually describe a TIBCO iProcess Objects *Director*, which is used to connect the client to a server). Select “true” if the specifications are for a Director, or “false” if a TIBCO iProcess Objects Director is not being used.
- **UserPreferencePersistence** - This element contains two attributes that are used to specify whether user data¹ is persisted locally or on the server, as well as obtained locally or from the server upon login. Server-side persistence allows users to move to different machines and/or browser types, and pick up user preferences specified from another machine and/or browser type. The user preference persistence attributes are:
 - persistOnServer** - This attribute specifies whether or not to persist on the server, as follows:
 - If “false”:
 - All user data is stored on the client.
 - User data is not cached and is persisted client-side immediately.
 - If “true”:
 - All user data is stored on the server.
 - User data is cached and is not saved to the server until the user logs out or closes the browser window.
 - Options values (i.e., all settings on the **Options** dialog in the application) are stored on both the client and the server. This is required because the language

¹User data consists of the following: Adding, removing, or changing views (note that changes to views are persisted immediately, whereas all other user data are persisted upon logout or application closure); list filters; list sorts; column changes (either using the Column Selector, or done manually); auto-repeat toggle on the work item list; case history show seconds/microseconds setting.

setting is stored in the Options data and this is needed to set up the locale before login.

The first time server-side data is accessed for a given user, the user is given the option of initializing the server-side data with any data that has previously been persisted client side (if any client-side data exists). The user's response to this question is persisted on the server and will not be asked again.

Default = "false" if attribute is absent.

maxDataSize - Sets the maximum number of bytes for the user preference data. This value needs to be set at or below the field size supported by the database used on the server, which is typically 256K (128K for double-byte character encoding).

If this value is too small, processing the data at the server will be inefficient; if it's too large, the database will throw an exception when it attempts to parse the message containing user preference data.

Note that the character encoding used should be taken into consideration when determining the maximum data size.

Default = 32768 (32K) if attribute is absent

Minimum value = 10

Additional records can be added if you would like more than three servers to appear in the **Server** field drop-down list.

Action Processor URL

When the client application is installed, you must specify the URL to the Action Processor to which you want the client application to connect when it is started. This URL is written to the client application's configuration file by the installation program.

You can later modify the Action Processor URL in the configuration file so that the client connects to a different Action Processor. You can also specify multiple URLs, and assign each a *weighting* value, which is used to determine the percentage of connections given to that URL (Action Processor). This allows load balancing of the available Action Processors. When the client application starts, it randomly selects (with weighting applied) one of the Action Processor URLs specified in the configuration file.

To configure the Action Processor URL:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).
2. Locate the **ActionProcessors** record in the `config.xml` file. For example:

```
<record jaxid="ActionProcessors" type="ipc">
  <ActionProcessor weighting="100"
    baseUrl="http://austin90/TIBCOActProc/ActionProcessor.servlet"/>
</record>
```

3. If you would like multiple Action Processors to be available for connections, add an additional **<ActionProcessor />** element for each Action Processor, then perform the following steps to configure each of the components of the **<ActionProcessor />** element. (The easiest method is to copy and paste the existing **<ActionProcessor />** element, then modify the **weighting** and **baseUrl** values according to the descriptions in steps 4 and 5.)

If you are just modifying the existing URL or weighting value, proceed to the following steps.

4. To specify the Action Processor to which the client should connect, modify the **baseUrl** attribute string to point to the desired Action Processor(s). This entry must be in the form:

```
http://Host:Port/APDir/ActionProcessor.ext
```

where:

- *Host* is the name of the machine hosting the Action Processor. (Note that if you are hosting both the client application and the Action Processor on the same machine, and they are both being hosted by Tomcat, you can specify *Host* as “localhost”.)

This can also be the IP address (either an IPv4 or IPv6 address) of the host machine. Note, however, if it is an IPv6 address, it must be enclosed in square brackets when in a URL. For example:

```
http://[2001:db8:85a3:8d3:1319:8a2e:370:7348]:port/restOfURL
```

- *Port* is the port number used by the Web Application Server (WAS) that is hosting the Action Processor to communicate with web applications.
- *APDir* is the directory on *Host* in which the Action Processor is installed.
- *ext* is the file name extension. This is “servlet” (for Java servlet) if you are connecting to a Java Action Processor, or “aspx” (for .NET ASP web application) if you are connecting to a .NET Action Processor.

The example shown in step 2 specifies that the client application connect to a Java Action Processor (hence the “servlet” extension) on machine “austin”. The WAS hosting the Action Processor is communicating with web applications on port 90, and the Action Processor was installed in the **TIBCOActProc** directory.

5. Specify a *weighting* value for each Action Processor by setting the **weighting** attribute as follows:
 - If you are only specifying a single URL, the **weighting** attribute can be set to any value, or it can be left unspecified.
 - For multiple URLs, the **weighting** value determines the percentage of connections based on the total of all **weighting** values. For instance, if a URL’s **weighting** value is 30% of the total of all **weighting** values (see the example below), the client will connect to it 30% of the time:

```
<record jsid="ActionProcessors" type="ipc">
  <ActionProcessor weighting="30"
    baseUrl="http://austin70/TIBCOActProc1/ActionProcessor.servlet"/>
  <ActionProcessor weighting="50"
    baseUrl="http://austin8500/TIBCOActProc2/ActionProcessor.servlet"/>
  <ActionProcessor weighting="20"
    baseUrl="http://austin9050/TIBCOActProc3/ActionProcessor.servlet"/>
</record>
```

Note that the weighting parameter is only for load balancing purposes — it is *not* to provide failover. If the Action Processor fails, the application should return to the **Login** screen.

The **weighting** values can total any number, although it’s easier to calculate the percentage for each if they total 100 as in the example above.

If an Action Processor is not available when the client attempts to connect to it, the weighting values are recalculated based on the remaining available Action Processors, and another URL is randomly selected. This process continues until a connection is made, or no active Action Processor can be found (in which case, an error is returned — the client application must be reloaded to continue).

Session Monitor

You can specify that if a user of the client application is inactive for a certain period of time, the user’s session will time out and automatically log the user out.

You can also specify when a warning dialog is to be displayed, informing the user that the session is about to time out. The user can click **OK** on this warning dialog to continue the session. If the user does not respond to the warning message, the session will time out in the specified period of time.



Note that if there is a timeout setting on the application server that is less than the setting specified using the **SessionMonitor** parameter, the setting on the application server takes precedence. See [Session Timeout](#).

To specify the session time-out:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).
2. Locate the **SessionMonitor** record in the `config.xml` file:

```
<record psid="SessionMonitor" timeout="30" warning="5" disable="false" type="ipc" />
```

3. Specify the record's attributes as follows:
 - **timeout** - The number of minutes of user inactivity before the session will time out. The user is automatically logged out upon timing out.
 Minimum: 5
 Maximum: none
 Default: 30
 - **warning** - The number of minutes *before* the time out will occur that a warning dialog is displayed informing the user that the session is about to time out.
 Minimum: 1
 Maximum: 1/3 of the value specified for the time-out period.
 Default: 5
 - **disable** - Set to "true" to disable session monitoring — the application will not time out; set to "false" to enable session monitoring.
 Default: false
4. Save and close the `config.xml` file.

Hide Case Data Tab Find Tool

The **CaseDataFind** configuration parameter allows you to remove the **Find** tool from the case **Data** tab.

The reason you would want to remove the **Find** tool is because of an issue that can cause the field list on the case **Data** tab to be empty. This issue can occur under the following circumstances:

- You are using Microsoft Internet Explorer (the issue does not occur when using Mozilla Firefox),
- the case contains memo fields that contain a large amount of XML data, and
- the **Find** tool on the case **Data** tab is enabled.

Under these circumstances, an XML transformation performed by the **Find** tool can cause the MSXML parser to fail, resulting in the empty field list.

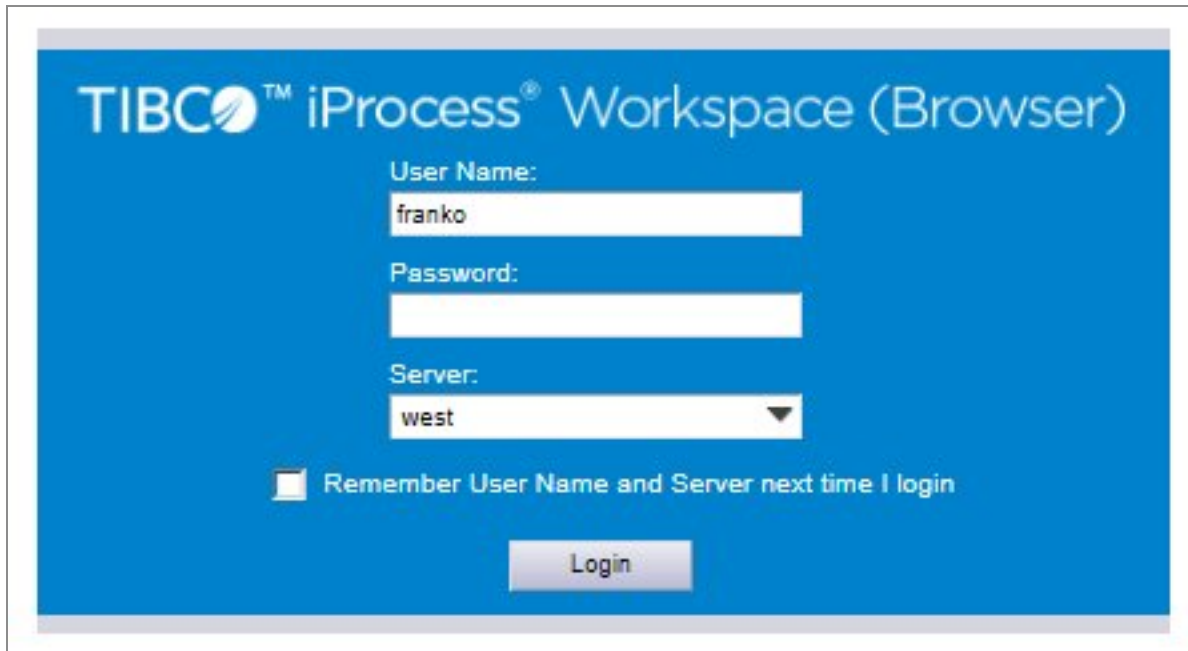
To specify whether or not to hide the **Find** tool:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).
2. Locate the **CaseDataFind** record in the `config.xml` file:

```
<record jsid="CaseDataFind" show="true" />
```
3. Set the **show** attribute as follows:
 - “true” - The **Find** tool is enabled on the case **Data** tab.
 - “false” - The **Find** tool is hidden on the case **Data** tab.
4. Save and close the `config.xml` file.

Remember Login Information

You can configure whether or not to display the **Remember User Id and Server next time I login** check box on the **Login** dialog:



By default, the check box is displayed.

To configure this option:

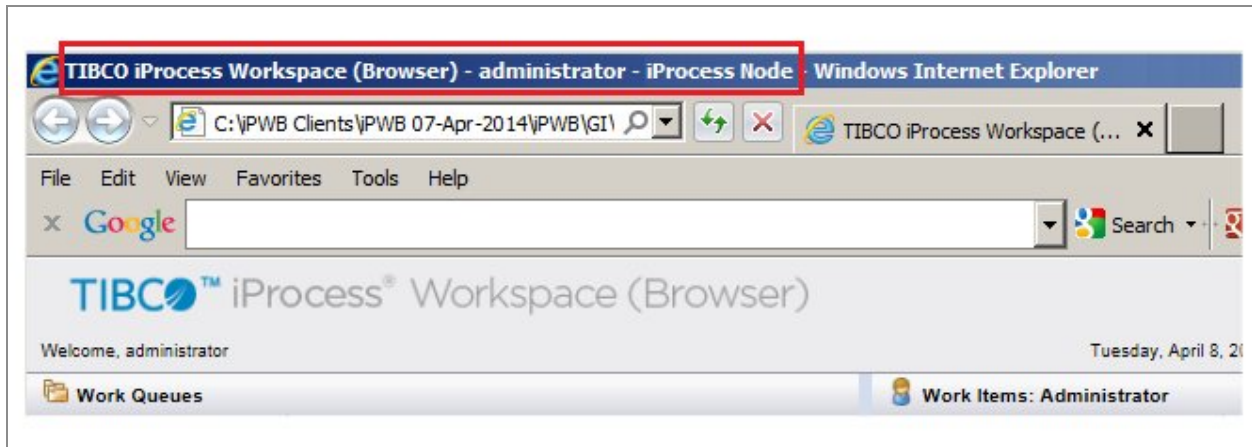
1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).
2. Locate the **Login** record in the `config.xml` file:

```
<record jaxid="Login" type="ipc" useRemember="true" allowDirectLogin="false"/>
```

3. Modify the **useRemember** attribute as follows:
 - “true” causes the check box to be displayed.
 - “false” causes the check box to not be displayed.

Customizing the Browser Window Caption

You can customize the caption that is displayed in the browser window after a user has logged into the client application.



By default, the caption is set to:

“TIBCO iProcess Workspace (Browser) - <Username> - <ServerNodeName>”

To customize the caption:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).

2. Locate the **postLoginCaption** record in the `config.xml` file:

```
<record jsid="postLoginCaption" pattern="%productname% - %username% - %serverNodeName%"/>

```

3. Modify the **pattern** attribute for the caption you would like displayed. The following placeholders can be used in the pattern string to display various information:

- **%productname%** - This placeholder is replaced with the name of the product.
- **%username%** - This placeholder is replaced with the name of the logged-in user.
- **%usernameDesc%** - This placeholder is replaced with the user's "DESCRIPTION" attribute defined on the server.
- **%serverNodeName%** - This placeholder is replaced with the node name of the server the user has logged into. Note that when single authentication is used (see [Single Authentication](#)), this value is available only if the logged-in user's access profile (see [User Access](#)) allows access to server information (**name="ServerInfo"**).
- **%displayName%** - This placeholder is replaced with the value of the **displayName** attribute of the server the user has logged into (see the **displayName** attribute for the **ServerNodes** element on [displayName: The name that you would like displayed in the iProcess Workspace \(Browser\)](#)

[Login to field drop-down list. This is the name the user would select when choosing a server to log into.](#)). If **displayNameName** is null, the server node name is displayed.

Any or all of the placeholders can be specified or omitted.

If a placeholder is specified that is not available, it will be replaced with a zero-length string.

Note that you can also customize the caption that is displayed in the window/dialog/preview pane for an opened work item — see [Customizing the Work Item Caption](#).

Customizing the Work Item Caption

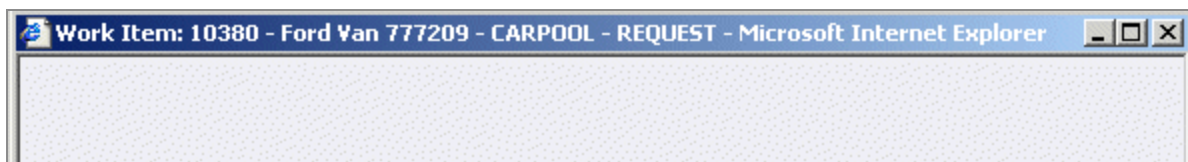
You can customize the caption that is displayed when you open a work item form.

Note that this applies only to work items that are opened from a work queue — it does not apply to the form displayed when starting a case. That caption always displays “Start Case: *CaseDescription* - *ProcedureName* - *StepName*”, which is the information known at that point.

Depending on how options are set for the display of work items, and the type of form used, the caption will be displayed in one of the following ways:

- in the title bar of a separate browser window, or
- as a dialog caption in the main window.

For example:



This illustration shows the default caption, which is defined using the **workItemCaption** parameter in the client’s `config.xml` file.

The following is the default caption:

Work Item: %caseNumber% - %caseDescription% - %procName% - %stepName%

Data from the work item will appear in place of the keywords shown.

To override the default setting:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).
2. Locate the **workItemCaption** record in the `config.xml` file:

```
<record jsxid="workItemCaption"  
  pattern="Work Item: %CaseNumber% - %Description% - %Proc_Name% - %StepName%" />
```

3. Remove the comment characters around the record (“<!--“ and “-->”).
4. Modify the **pattern** attribute for the caption you would like displayed. A list of the placeholders that can be used in the record appears in the comments above the setting in `config.xml`.

**Note**

The default caption uses localized text. When overriding the setting in this way localized text is no longer used; the setting applies to all languages.

Note that you can also customize the caption that is displayed in the browser window in which the client application is running — see [Customizing the Browser Window Caption](#).

**Note**

If you are using GI Forms in your application, you can customize the form caption on an individual-form basis. This is done by setting the **caption** property in your `FormTemplate.js` file to the customized value. The same data placeholders can be used that are used for the **workItemCaption** parameter. The value specified in the **caption** property overrides the **workItemCaption** parameter in the `config.xml` file. For more information, see the comments in the `FormTemplate.js` file.

Specifying Browser Window Features

You can customize the appearance of the browser window when displaying work item forms. You can specify things such as whether the window is resizable, whether or not a status bar is displayed, etc.

Note that the extent to which you can customize your browser window appearance depends on the type of browser (Internet Explorer or Firefox) you are using.

To configure browser features for your work item forms:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).
2. Locate the **BrowserFeatures** record in the `config.xml` file:

```
<record jsxid="BrowserFeatures" type="ipc">
  <ExternalForms channelmode="no" dialog="no"
directories="no"
      location="no" menubar="no" minimizable="no"
resizable="yes"
      status="yes" toolbar="no"/>
  <GIForms channelmode="no" dialog="no" directories="no"
location="no"
      menubar="no" minimizable="no" resizable="yes"
scrollbars="yes"
      status="yes" toolbar="no"/>
</record>
```

3. Modify the desired attributes for the appropriate form category. Each attribute can be set to either “yes” or “no” to indicate whether or not to display/enable that feature.
 - For information about the different form categories, see [Form Type](#).
 - For information about the available attributes, see [Browser Feature Attributes](#).

Form Type

Browser features are specified separately for each of the following categories of work item forms:

- **External Forms** - This category includes the following types of forms:
 - ASP Forms
 - JSP Forms
 - BusinessWorks FormBuilder Forms

The **<ExternalForms>** record attributes are used to control the browser features for external forms.

- **GI Forms** - This category includes the following types of forms:

- TIBCO® General Interface Forms
- TIBCO Forms

The **<GIForms>** record attributes are used to control the browser features for GI Forms.

```
<record jsxid="BrowserFeatures" type="ipc">
  <ExternalForms channelmode="no" dialog="no"
directories="no"
      location="no" menubar="no" minimizable="no"
resizable="yes"
      status="yes" toolbar="no"/>
  <GIForms channelmode="no" dialog="no" directories="no"
location="no"
      menubar="no" minimizable="no" resizable="yes"
scrollbars="yes"
      status="yes" toolbar="no"/>
</record>
```



Note

Note that iProcess Modeler forms are also considered external forms. However, browser features cannot be used with TIBCO iProcess Modeler-produced work item forms.

Browser Feature Attributes

The following table lists the available attributes for the **<BrowserFeatures>** record. The **Browsers** columns indicate the browsers to which the attribute applies.

Attribute	Browsers		Description
	Internet Explorer	Firefox	
channelmode	X		Specifies whether or not to display the window in “theater mode”, that is, as a maximized window. When set to "yes", the height, width, top and left values are overridden, the Navigation Bar is hidden, and the Title Bar is visible.
dialog	X	X	Specifies whether or not to display the window as a “dialog”. • If the WCC/client application is being run locally, and you are using Internet Explorer, the "dialog" attribute must be set to "no" — it can be set to "yes" only if the application is run from a web server. (If you are using Firefox, the "dialog" attribute can be set to "yes" or "no", even if you are running locally.) • If "dialog" is set to "yes", it behaves differently for external forms than for GI forms, as follows: — For external forms: if “dialog” = yes, the form opens in a “Webpage” dialog. For information about webpage dialogs, see Dialog/Window Characteristics. — For GI forms: if “dialog” = yes, the form opens in an “application” dialog. For information about application dialogs, see Dialog/Window Characteristics. Note - For information about external

Attribute	Browsers		Description
	Internet Explorer	Firefox	
			forms and GI forms, see Form Type. If “dialog” = no, the form opens in a separate browser window (this is true for both external forms and GI forms). For information about separate browser windows, see Dialog/Window Characteristics.
directories		X	Specifies whether or not to display the "Personal Toolbar" and the "Bookmarks Toolbar" in Firefox. Firefox users can force new windows to always render the Personal Toolbar/Bookmarks Toolbar by setting <code>dom.disable_window_open_feature.directories</code> to true in about:config¹ or in their user.js file.
location	X	X	Specifies whether or not to display the "Navigation Toolbar" in IE or the "Location Bar" in Firefox. Firefox users can force new windows to always render the Location Bar by setting <code>dom.disable_window_open_feature.location</code> to true in about:config or in their user.js file.
menubar	X	X	Specifies whether or not the browser window should display a “Menu Bar”. Firefox users can force new windows to always

¹Firefox can be configured by entering “about:config” in the Firefox address bar.

Attribute	Browsers		Description
	Internet Explorer	Firefox	
			render the Menu Bar by setting <code>dom.disable_window_open_feature.menubar</code> to true in about:config or in their user.js file.
minimizable		X	Specifies whether or not to allow the browser window to be minimized. This is only applicable when dialog=”yes”, which causes the Maximize and Minimize buttons to <i>not</i> be displayed. Setting minimizable=yes causes both the Maximize and Minimize buttons to be displayed, but the Minimize button is enabled and the Maximize button is disabled. Also see the dialog attribute above.
resizable	X		Specifies whether or not the browser window can be manually resized using the lower right corner of the window. Firefox always makes windows resizable. Note - This attribute may not work as expected. Tests on various systems has shown that on some the window can be resized, while on others, it cannot. The exact cause(s) of the unexpected behavior remains unknown, although it is thought to be a combination of the browser being used, the browser version, and browser security settings.
scrollbars	X	X	Controls the display of scrollbars, on work item forms opened in a separate browser window, when the form content overflows the browser dimensions.

Attribute	Browsers		Description
	Internet Explorer	Firefox	
			Note that this only applies to GI Forms. When using external forms (ASP, JSP, BusinessWorks FormBuilder, iProcess Modeler), scrollbars will always appear if the form content exceeds the browser dimensions.
status	X		Specifies whether or not the browser window displays a status bar on the bottom of the window. Firefox always displays the status bar.
toolbar	X	X	Specifies whether or not to display the Toolbar across the top of the window. This bar contains buttons/icons for Back, Forward, Refresh, Home, etc. In IE, this bar is referred to as the “Command Bar”; and in Firefox, the “Tab Bar”. Firefox users can force new windows to always render the Tab Bar by setting <code>dom.disable_window_open_feature.toolbar</code> to true in about:config or in their user.js file.

Dialog/Window Characteristics

When a WCC or client application displays a work item form, it displays it either in a preview pane, in a separate dialog, or in a separate browser window. You can choose which of these formats you want from within the application (for more information, see the *TIBCO iProcess Workspace (Browser) User's Guide*).

Note, however, the type of form you are using determines which of the form formats (preview pane, dialog, or separate browser window) are selectable from the application, as follows:

- if your application uses GI forms, you can choose to open them in any of the three available formats: Preview Pane, dialog, or separate browser window.

- if your application uses external forms, they will always be opened in a separate browser window.

Also note that “dialogs” are further subdivided into the following:

- Webpage dialogs
- Application dialogs

Whether the work item form opens in a “Webpage” dialog or an “application” dialog depends on the setting of the “dialog” attribute in the <**BrowserFeatures**> record in the application’s `config.xml` file. For more information, see the “dialog” attribute description on [dialog](#).

The following describes the differences in behavior between the different types of dialogs/windows:

- **Minimize/Maximize Buttons** - Webpage dialogs do not have minimize nor maximize buttons. Separate browser windows and application dialogs have these buttons.
- **Floating Window Outside Application Window** - Both Webpage dialogs and separate browser windows can be floated outside the parent application's window, whereas application dialogs cannot.
- **Browser Feature Attributes** - The Browser Feature attributes (i.e., the attributes of the <**BrowserFeatures**> record in the `config.xml` file) supported depends on the dialog/window and the type of browser used, as follows:
 - **Webpage dialog:** If using Internet Explorer, only the "resizable" and "status" attributes are supported. If using Firefox, the supported attributes are: "dialog", "directories", "location", "menubar", "minimizable", and "toolbar".
 - **Application dialog:** None of the Browser Feature attributes are supported for this type of dialog.
 - **Separate browser window:** The table on the preceding pages lists the browser features that are supported for each of the available browsers.
- **Close as child window:** Both Webpage dialogs and application dialogs are children of the parent window, therefore if the parent window is closed (or minimized), the Webpage/application dialog is also closed (or minimized). Separate browser windows do not close (or minimize) when the parent is closed (or minimized).

Releasing Resources on Logout

You can specify the level that resources are released when a user logs out of the client application.

To configure this option:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).

2. Locate the **Logout** record in the `config.xml` file:

```
<record joid="Logout" releaseAllResources="true"
  logoutUrl=""
  timeoutUrl="">
```

3. Modify the **releaseAllResources** attribute as follows:

- “false” causes the “user session” to be closed, which closes the TCP connection between the client and the iProcess Objects Server.
- “true” (the default) causes the “user session” to be closed, which closes the TCP connection between the client and the iProcess Objects Server. This also closes the “SAL session”, which releases all lists held on the iProcess Objects Server
- this releases all client-side and server-side resources.

Redirecting Client to URL on Logout

You can specify that the client be redirected to a specified URL when the user logs out.

Note that the **logoutUrl** parameter applies to standard logins/logouts, as well as when single authentication is used to log into the application.

To configure this option:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).

2. Locate the **Logout** record in the `config.xml` file:

```
<record joid="Logout" releaseAllResources="true"
  logoutUrl=""
  timeoutUrl="">
```

3. Specify the desired URL in the **logoutUrl** attribute string.

If an empty string is specified, the client is redirected to the **Login** screen.

You can also specify “close.html” in the **logoutUrl** attribute string to cause the browser window to close upon user logout.

**Note**

Not all browsers allow a web application to close the main browser window in which it is running. Therefore, if **logoutUrl** is used to redirect to `close.html`, when the redirection occurs some browsers will leave the window open to a blank page. This can be observed in recent versions of Firefox, where the behavior is intentional. Since the main window was opened by the user and not by the web application, the browser does not allow the web application to close it.

Redirecting Client to URL on Browser Session Timeout

You can specify that the client be redirected to a specified URL upon a browser session time out.

To configure this option:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).
2. Locate the **Logout** record in the `config.xml` file:

```
<record jsid="Logout" releaseAllResources="true"
logoutUrl=""
timeoutUrl="">
```

3. Specify the desired URL in the **timeoutUrl** attribute string.

If an empty string is specified, the client is redirected to the **Login** screen. (If single authentication is being used, and the **timeoutUrl** attribute is an empty string, the browser window closes upon a session timeout.)

You can also specify “close.html” in the **timeoutUrl** attribute string to cause the browser window to close if a timeout occurs.



Not all browsers allow a web application to close the main browser window in which it is running. Therefore, if **timeoutUrl** is used to redirect to `close.html`, when the redirection occurs some browsers will leave the window open to a blank page. This can be observed in recent versions of Firefox, where the behavior is intentional. Since the main window was opened by the user and not by the web application, the browser does not allow the web application to close it.

User Options

The client application contains an **Options** dialog from which each user can specify their personal *user options*. User options establish default settings for each user who logs into the client application. These include things such as the list (procedure or work queue) to display first, the size and location of forms, the language to display, etc. For information about setting user options from the **Options** dialog, see the *iProcess Workspace (Browser) User's Guide*.

There are *default* user options defined in the system that each new user inherits until they specify their own user options on the **Options** dialog. These default user options are defined in the client application's configuration file, `config.xml`.

Note that the **Options** dialog in the client application also contains a **Defaults** button that sets all of the options for the user to the default user options specified in the `config.xml` file.

To configure the default user options:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).
2. Locate the **Options** record:

```
<record jsid="Options" type="ipc">
  <options>
    <display localKey="m_US" initialDisplay="workQ" ...
    <filter filterCases="specify" thresholdCases="500" ...
    <layout previewCases="on" previewWorkItems="off" ...
    <outstanding recurse="true"></outstanding>
    <subcase precedence="swPrecedenceR"></subcase>
```

3. Using the information in the following table, change the values of the appropriate element attributes to the desired default values:

Element	Attribute	Possible Values	Meaning
<display />	localeKey	Locale key specified in <i>ClientInstallDir\JSXAPPS\ipc\locale\locales.xml</i> .	Specifies the language in which the client application is displayed. (For more information, see Localization.)
	initialDisplay	procs	Displays the procedure list upon login.
		workQs	Displays the work queue list upon login.
	captionCases	name	Causes the case name to be displayed in the case list caption.
		description	Causes the case description to be displayed in the case list caption.
	captionWorkItems	name	Causes the work queue name to be displayed in the work item list caption.
		description	Causes the work queue description to be displayed in the work item list caption.
	autoRefreshWorkItems	true	Causes work item lists to be automatically refreshed.
		false	Work item lists are not automatically refreshed.
	autoRefreshInterval	integer value	Number of seconds between automatic refreshes of work item lists when

Element	Attribute	Possible Values	Meaning
	autoRefreshApplyAll		autoRefreshWorkItems = true.
		true	Changes to auto-refresh settings via the Options dialog, apply to currently open lists, as well as lists opened in the future.
		false	Changes to auto-refresh settings via the Options dialog, apply only to lists opened in the future.
<filter />	filterCases	always	The Filter dialog is always displayed when you open a case list.
		never	The Filter dialog is never displayed when you open a case list (you can manually display the Filter dialog).
		specify	This is used in conjunction with the thresholdCases attribute. If this value is specified, the Filter dialog is automatically displayed if the number of cases of the selected procedure exceeds the number in the thresholdCases attribute. If the number of cases does not exceed the threshold, the case list is displayed without displaying the Filter dialog.

Element	Attribute	Possible Values	Meaning
	thresholdCases	integer value	See the specify value for the filterCases attribute above.
	forceCasesThreshold	true	Forces the user to filter cases until the number of cases is below the threshold set in thresholdCases . The filter dialog is displayed, and the user cannot view the list until a filter is specified that brings the number of cases below the threshold. This is applicable only when filterCases is set to “specify”.
		false	The user is not forced to filter the number of cases below the threshold set in thresholdCases .
	<filter /> (Cont.)	always	The Filter dialog is always displayed when you open a work item list.
		never	The Filter dialog is never displayed when you open a work item list (you can manually display the Filter dialog). The first page of the work item list is always displayed. Use caution with this selection as it can have a negative impact on performance if the page size is set to a large value.

Element	Attribute	Possible Values	Meaning
		specify	This is used in conjunction with the thresholdWorkItems attribute. If this value is specified, the first page of the work item list is downloaded from the iProcess Engine when you open a work queue from the work queue list only if the number of work items does not exceed the number in the thresholdWorkItems attribute. If the number exceeds the threshold, the Filter dialog is displayed first, allowing you to apply a filter so a large number of work items won't be downloaded.
	thresholdWorkItems	integer value	See the specify value for the filterWorkItems attribute above.
<filter /> (Cont.)	forceWorkItemsThreshold	true	Forces the user to filter work items until the number of work items is below the threshold set in thresholdWorkItems . The filter dialog is displayed, and the user cannot view the list until a filter is specified that brings the number of work items below the threshold. This is applicable only when filterWorkItems is set to "specify".

Element	Attribute	Possible Values	Meaning
		false	The user is not forced to filter the number of work items below the threshold set in thresholdWorkItems .
<layout />	previewCase	on	The case summary is shown in the Preview Pane when a case is selected; case details are shown in the Preview Pane when a case is opened.
<layout /> (Cont.)	previewCase (Cont.)	float	The case summary is shown in the Preview Pane when a case is selected; case details are shown in a floating window when a case is opened.
		off	The Preview Pane is turned off; no display when a case is selected; case details are shown in a floating window when a case is opened.
	previewWorkItems	on	The work Item summary is shown in the Preview Pane when a work item is selected; a work item form is shown in the Preview Pane when a work item is opened.
		float	The work Item summary is shown in the Preview Pane when a work item is selected; a work item form is shown in a floating window when a work item is opened.

Element	Attribute	Possible Values	Meaning
		off	The Preview Pane is turned off; no display when a work item is selected; a work item form is shown in a floating window when a work item is opened.
	previewCaseResize	false	The Preview Pane is not resized when a case is opened in the Preview Pane.
		true	The Preview Pane is resized to the percentage specified in the previewCaseSize attribute when a case is opened in the Preview Pane. It reverts to the previous size when the case details are closed.
	previewWorkItemResize	false	The Preview Pane is not resized when a work item is opened in the Preview Pane.
<layout /> (Cont.)	previewWorkItemResize (Cont.)	true	The Preview Pane is resized to the percentage specified in the previewWorkItemSize attribute when a work item is opened in the Preview Pane. It reverts to the previous size when it is closed.
	previewCaseSize	integer value	The percentage (from 1 - 100) of the viewing area the Preview Pane should encompass when automatically resizing the

Element	Attribute	Possible Values	Meaning
			Preview Pane when a case is opened in the Preview Pane (see the previewCaseResize attribute).
	previewWorkItemSize	integer value	The percentage (from 1 - 100) of the viewing area the Preview Pane should encompass when automatically resizing the Preview Pane when a work item is opened in the Preview Pane (see the previewWorkItemResize attribute).
	floatWorkItems	dialog	Floating windows containing a work item form are displayed in a separate dialog.
		browser	Floating windows containing a work item form are displayed in a separate browser window.
	modalDialog	false	Work item forms are not modal.
		true	Work item forms are displayed modal (i.e., the user cannot perform any other functions until that dialog is closed).
	floatLeft	integer value	The floating window is positioned this number of

Element	Attribute	Possible Values	Meaning
			pixels from the left. (Only applicable if both the floatFullscreen and floatCenter attributes are false.)
<layout /> (Cont.)	floatTop	integer value	The floating window is positioned this number of pixels from the top. (Only applicable if both the floatFullscreen and floatCenter attributes are false.)
	floatWidth	integer value	The width (in pixels) of the floating window. (Only applicable if the floatFullscreen attribute is false.)
	floatHeight	integer value	The height (in pixels) of the floating window. (Only applicable if the floatFullscreen attribute is false.)
	floatFullscreen	true	The floating window is displayed full screen.
		false	The floating window is not displayed full screen. (Use other attributes to determine position/size.)
	floatCenter	true	The floating window is displayed centered. (Use floatWidth and floatHeight

Element	Attribute	Possible Values	Meaning
			attributes to determine size.)
		false	The floating window is not displayed centered. (Use other attributes to determine position/size.)
	floatRememberPostion	true	The system remembers the size and position of the floating window if you manually move it on your screen.
		false	Future floating windows are opened in the position and size specified by the other <layout /> element <i>float</i> attributes.
<outstanding />	recurse	true	Causes the default setting of the Recurse sub-cases for outstanding items check box on lists of outstanding work items to be <i>checked</i> . (This check box determines whether or not the list should include work items in sub-cases.)
		false	Causes the default setting of the Recurse sub-cases for outstanding items check box on lists of outstanding work items to be <i>unchecked</i> .

Element	Attribute	Possible Values	Meaning
<subcase />	precedence	swPrecedenceR	The precedence in which sub-procedures are started from the main procedure:
		swPrecedenceUR	— swPrecedenceR: Only released sub-procedures are started.
		swPrecedenceMR	— swPrecedenceUR: Unreleased, then released.
		swPrecedenceUMR	— swPrecedenceMR: Model, then released.
		swPrecedenceMUR	— swPrecedenceUMR: Unreleased, model, then released.
			— swPrecedenceMUR: Model, unreleased, then released.

Limiting Number of Cases

By default, when a case list is displayed, all available cases for the selected procedure are downloaded from the server. If there is a very large number of cases for the selected procedure, the list may be slow in displaying.

For this reason, the **MaxCases** parameter is available to limit the number of cases to download from the server. If this parameter is set in the application's `config.xml` file, when a user selects a procedure in the application, and the number of cases for that procedure exceeds the number specified, the number downloaded is limited to the number specified.

When the number of cases downloaded is limited by this parameter, a message is displayed on the bottom of the case list informing the user of the number of cases that were downloaded, as well as the number that were not downloaded because they exceeded the maximum number specified.

To limit the number of cases to download:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).
2. Locate the **MaxCases** record in the `config.xml` file. For example:

```
<record joid="MaxCases" maximum="" />
```

3. Enter the desired number in the quotes of the **maximum** attribute:

```
<record joid="MaxCases" maximum="100" />
```

If **maximum** is set to an empty string or a non-positive number, all available cases are downloaded when the user selects a procedure.

Setting the Maximum Number of Case History Entries

If the number of entries in the case history list is very large (e.g., greater than 5000), responsiveness of the list is degraded, and the chance of an out-of-memory condition is increased.

For this reason, there is configuration parameter that limits the number of case history entries to a specified number.

If the requested number of entries exceeds this specified number, the list is truncated to the maximum, the count of excluded items is displayed, and the user is instructed to filter the list.

To configure this parameter:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).

2. Locate the **MaxHistory** record in the `config.xml` file:

```
<record joid="MaxHistory" maximum="5000"/>
```

3. In the **maximum** attribute string, specify the maximum number of entries for the case history list.

Default = "5000" (which is imposed if **maximum** is empty or a negative value)

Minimum = "1"

Maximum = "5000"

Specifying Default Page Size for Work Item Lists

You can specify the default number of work items to display in a work item list.

The user can modify this value using the **Page Size** function on the work item list (access to the Page Size function can also be control via a user access profile property).

To configure this parameter:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).
2. Locate the **PageSize** record in the `config.xml` file:

```
<record joid="PageSize" default="20"/>
```

3. In the **default** attribute string, specify the desired number of work items you would like displayed in the work item list by default.

Specifying Default Types/Statuses to Display on Lists

You can specify the default types and statuses of procedures and work queues to display in the procedure and work queue list. These include:

- Procedure statuses on the procedure list:
 - Released procedures
 - Unreleased procedures

- Model procedures
- Withdrawn procedures
- Procedure types on the procedure list:
 - Main procedures
 - Sub-procedures
 - Main and sub-procedures
- Work queue statuses on the work queue list:
 - Released work queues
 - Test work queues
 - Released and test work queues
- Work queue types on the work queue list:
 - User work queues
 - Group work queues
 - User and group work queues

Note that setting the configuration parameters for these only specifies what is displayed *by default*. The user can change the statuses/types to display by making the desired selections from the **View** menu on the procedure or work item list.

To specify default types/statuses:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).
2. Locate the **ListView** record in the `config.xml` file:

```
<record jaxid="ListView" type="ipc">
  <procedureStatus released="true" unreleased="false" model="false" withdrawn="false">
    </procedureStatus>
    <procedureType main="true" sub="true"></procedureType>
    <workQueueStatus released="true" test="false"></workQueueStatus>
  </record>
```

3. Set the applicable attributes to “true” or “false” to control whether or not that status/type is displayed by default:
 - “true” causes that type/status to be displayed on the applicable list by default. The associated selection on the **View** menu will also show a check mark, indicating that that type/status is currently displayed.

- “false” causes that type/status to not be displayed on the applicable list by default (although the user can display that type/status by selecting it from the **View** menu).

Server-Side Atomic Locking of Work Items

To ensure that only available work items are locked when using the “Open First Available Work Item” or “Open Next Available Work Item” functions, the TIBCO iProcess Workspace (Browser) supports *server-side atomic locking of work items*.

Server-side atomic locking of work items results in the selection of the work item to lock to occur on the server (instead of on the client) when using the “Open First Available Work Item” and “Open Next Available Work Item” functions.

If the server-side atomic locking feature is not used, and the user opens work items using the aforementioned functions, an error can appear because the work item to open is being selected from the work item list on the client, which is a “snap shot” and doesn't reflect locks that have occurred by other users on work items in that snap shot — the selected work item may not be available anymore.

The **AtomicServerLock** parameter is used to specify whether or not work items are locked at the server or on the client when using the functions mentioned above.



Note

Note that to use this “atomic server lock” feature, your TIBCO iProcess Objects Server must have MR 38404 implemented. Also note that if your TIBCO iProcess Objects Server also contains MR 41569, the operation of this feature differs, as described below.

To configure this parameter:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).

2. Locate the **AtomicServerLock** record in the `config.xml` file:

```
<record jsid="AtomicServerLock" supported="false" refresh="false"/>
```

3. Set the values in the **supported** and **refresh** attributes to “false” or “true” according to the descriptions below.

‘supported’ attribute

- If set to “true”, the work item selection for the aforementioned functions occurs on the server.
- If set to “false”, the work item selection occurs on the client. If you set **supported** = “false”, a “Work item already locked” error may occur when the “Open First Available Work Item” or “Open Next Available Work Item” function is used. (This provides backward compatibility for TIBCO iProcess Object Servers that don't have MR 38404.)

‘refresh’ attribute

- If set to “true”, the snap shot of the work items is refreshed prior to the TIBCO iProcess Workspace (Browser) requesting that the server atomically lock the work item.
- If set to “false”, the snap shot of the work items is not refreshed prior to the TIBCO iProcess Workspace (Browser) requesting that the server atomically lock the work item.

The following lists the reasons you may want to set the **refresh** attribute to “true” or “false” when using the atomic server lock feature (i.e., **supported** = “true”), depending on whether your TIBCO iProcess Objects Server has only MR 38404, or both MR 38404 and MR 41569:

- Your TIBCO iProcess Objects Server has only MR 38404:
 - If you are only concerned with ensuring that the server atomically lock the first available work item in the snap shot for the “Open First Available Work Item” and “Open Next Available Work Item” functions, set the attributes as follows:
 - supported = “true”
 - refresh = “false”
 - If you want to ensure that the atomic server lock includes new work items, and excludes work items that would no longer be in the work queue because they no longer match the filter, set the attributes as follows:
 - supported = “true”
 - refresh = “true”
- Your TIBCO iProcess Objects Server has both MR 38404 and MR 41569:

- If you are not concerned that the atomic server lock includes new work items that have arrived in the work queue since the last refresh, set the attributes as follows:
 - supported = “true”
 - refresh = “false”
- If you also want to ensure that the snap shot be refreshed to include new work items, before the server atomically locks a work item when using the aforementioned functions, set the attributes as follows:
 - supported = “true”
 - refresh = “true”

Note that when your server has MR 41569, the server also re-applies the filter that was specified on the work queue, to the snap shot, prior to selecting the work item. Therefore, work items in the snap shot that would no longer match the filter are excluded by the atomic server lock.

Additional notes concerning the “atomic server lock” feature:

- There is a cost to setting **refresh** = “true” — the work item list is refreshed every time the user selects the “Open First Available Work Item” or “Open Next Available Work Item” function. Therefore, you should set **refresh** = “true” only if a refresh is needed.
- Your TIBCO iProcess Engine's “WIS_QCHANGE_EXTENDED_CHECK” process attribute must be set to 1.
- If you are using a TIBCO iProcess Objects Server that does not support the “atomic server lock” feature (i.e., it does not have MR 38404), and a user attempts to use the “Open First Available Work Item” or “Open Next Available Work Item” function, a “Lock First Work Item not supported” message is displayed.

Specifying Whether Case Counts Should be Obtained

The procedure list may be configured (using the Column Selector) to display counts of active cases, total cases, and closed cases for each procedure. Systems with very large

numbers of procedures and cases may wish to avoid the overhead involved with obtaining these case counts.

For this reason, the **CaseCounts** parameter is available to be able to specify whether or not case counts are calculated and obtained.

To specify whether or not to obtain case counts:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).
2. Locate the **CaseCounts** record in the `config.xml` file. For example:

```
<record jidid="CaseCounts" show="true"/>
```

3. To obtain case counts, ensure the **show** attribute is set to “true”; to specify that case counts not be obtained, set the **show** attribute to “false”.

Note that if the Active Cases, Closed Cases, or Total Cases columns are displayed in the procedure list, and you have set the **show** attribute to “false”, the column headers are displayed with a line through them, and there are no counts in the columns.

Also note that if you are not getting case counts, but you are displaying the case count columns, if the Column Selector is opened and you change any columns in the procedure list, the Active Cases, Closed Cases, and Total Cases column headers will no longer have a line through them; refreshing the list causes them to be re-displayed. (Although, typically, if you are not getting case counts, you will not be displaying the case count columns.)

Specifying Outstanding Work Item Step Types

The **OutstandingTypes** configuration parameter is available to allow you to specify the types of steps that will appear in the list of outstanding work items for a case.

To specify step types for the list of outstanding work items:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).
2. Locate the **OutstandingTypes** record in the `config.xml` file. For example:

```
<record jidid="OutstandingTypes"
  includeNormalSteps="true"
  includeEventSteps="true"
  includeEASSteps="true"
  includeSubProcSteps="true"
  includeDynamicSubProcSteps="true"
  includeGraftSteps="true"
  includeTransactionControlSteps="true"/>
```


- For each of the step types listed in the attributes of the **OutstandingTypes** record, specify either “true” to include that type in the list of outstanding work items, or “false” to exclude that step type.

WebDAV Root Setting

If you are using TIBCO Forms, the base URL of the form’s location must be specified in the **webDAVRoot** parameter in the application’s `config.xml` file.



Web-based Distributed Authoring and Versioning (WebDAV) is a protocol used for publishing and managing content to web servers. TIBCO Forms uses WebDAV to publish forms.

When the client application is installed, the installation program asks if TIBCO Forms are being used, and if so, it allows you to enter the root to WebDAV, which the installation program then writes to the **webDAVRoot** parameter in `config.xml`.

If the person installing the client application does not enter the root to WebDAV at that time, it can be specified later by editing the **webDAVRoot** parameter in `config.xml`.

To specify the WebDAV root for TIBCO Forms:

- Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file’s location, see [Configuration Files](#).
- Locate the **webDAVRoot** record in the `config.xml` file. For example:

```
<record jxid="webDAVRoot">
```

- Replace `%TIBCO_CLIENT_WEBDAVROOT%` with the base URL of the location at which the TIBCO Forms are stored (this will already contain a URL if one had been entered during the installation). For example:

```
<record jxid="webDAVRoot" URL="http://myserver8090/webDAV"/>
```

For more information about TIBCO Forms, see [Forms](#).



The iProcess Workspace (Browser) software automatically loads a JavaScript file called `webDAVRoot/META-INF/form_ext.js`. This file allows loading and using custom JavaScript libraries inside TIBCO Form validations and events. The `form_ext.js` file is customized and deployed from TIBCO Business Studio. For more information, see the *TIBCO Business Studio Forms User's Guide*.

Add-ins

The **addins** parameter contains entries for TIBCO add-ins that are used by the client application and/or TIBCO® General Interface Builder when creating a custom WCC application, as described below:

```
<record jsid="addins" type="array">
  <record jsid="0" type="string">userforms</record>
  <record jsid="1" type="string">userforms2</record>
  <record jsid="2" type="string">wcc</record>
```

The **addins** parameter may contain the following entries:

- **user:forms** - This add-in must be specified in the `config.xml` file if the client application is using TIBCO Forms version 1.1. This add-in is used at runtime to display version 1.1 TIBCO Forms.
- **user:forms2** - This add-in must be specified in the `config.xml` file if the client application is using TIBCO Forms version 2.x. This add-in is used at runtime to display version 2.x TIBCO Forms.
- **wcc** - This add-in is used only in the `config.xml` file for custom WCC applications. It provides access to the WCC components in TIBCO GI Builder during design time.

You do not need to explicitly add any of these add-ins. All three of them appear in the `config.xml` file for custom WCC applications by default. Only the forms add-ins appear in the TIBCO iProcess Workspace (Browser) client application `config.xml` file by default, as that application does not use the `wcc` add-in.

TIBCO Forms Caching

The **formsConfig** parameter is used to specify caching of TIBCO Forms. Setting the **ignoreCache** attribute to true causes TIBCO Forms and their resources to be reloaded on

each request instead of retrieving the resources from the browser cache.

To set TIBCO Forms caching:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).
2. Locate the **formsConfig** record in the `config.xml` file. For example:

```
<record jpsid="formsConfig" ignoreCache="false"/>
```

3. To cause TIBCO Forms to reload on each request rather than be loaded from cache, set the **ignoreCache** attribute to true:

```
<record jpsid="formsConfig" ignoreCache="true"/>
```

Show/Hide Personal Work Queues

The **personalWorkQueue** configuration parameter specifies whether or not the logged-in user's personal work queue is shown or hidden by default.

This parameter can be used in conjunction with the **<WorkQueue>** / **<PersonalWorkQueueOption>** user access profile property in the `userAccessProfiles.xml` file, which specifies whether or not the user can change the default specified in this parameter. For more information, see [WorkQueue](#) [PersonalWorkQueueOption](#).

To specify whether to show or hide the logged-in user's personal work queue by default:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).
2. Locate the **personalWorkQueue** record in the `config.xml` file. For example:

```
<record jpsid="personalWorkQueue" default="hide"/>
```

3. Set the **default** attribute as follows:
 - To show the queue by default, set **default = "show"**.
 - To hide the queue by default, set **default = "hide"**.

Refresh Work Item List After Work Item is Opened Externally

The **externalLock** configuration parameter specifies a delay (in milliseconds) after which the work item list is automatically refreshed when an external form is opened.

When external work item forms (ASP, JSP, iProcess Modeler Forms, etc.) are opened, the work item is locked by an external process. Therefore, the work item list does not reflect the locked state of the work item because it was performed externally. The time delay specified in this parameter causes the work item list to be refreshed after a work item is opened externally. The delay provides sufficient time for the external lock of the work item to take place before the list is updated to reflect the locked state:

To specify the delay after which the work item list is automatically refreshed when an external form is opened:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file's location, see [Configuration Files](#).
2. Locate the **externalLock** record in the `config.xml` file. For example:

```
<record joid="externalLock" refreshDelay="500"/>
```

3. Set the **refreshDelay** attribute to the number of milliseconds to wait after an external form is opened before refreshing the work item list.

Customizations

This section describes customization tasks that can be performed on your application, either the iProcess Client or a custom WCC application.

Font and Image Settings

A cascading style sheet-like file is provided that allows you to customize the appearance of the client application. This file can be modified to change elements such as:

- icon/button images
- font type, size, color
- background colors

The appearance of these elements is defined in the following file:

```
ClientInstallDir\JSXAPPS\ipc\jss\ipcCSS.xml
```

An excerpt from this file is shown below:

```
<!-- background color for lists -->  
<record jsxid="ipcList BGC" type="jsxbgcolor" jsxttext="#ffffeff" />  
<!-- background effect for menu bars -->  
<record jsxid="ipcMenu BG" type="jsxbg" jsxttext="#ececee" />
```

This file is well-commented, making it easy to find the area you would like to customize. To make a change to a font, image, color, etc., modify the value in the **jsxttext** attribute.

Adding Custom Menu Items and Toolbar Buttons

Custom menus and/or toolbar buttons can be added to the client application using configuration settings in the client application's configuration file, `config.xml`.

The `record` element that specifies a custom menu or toolbar button has a **jsxid** attribute value of **customMenus**:

```
<record jsxid="customMenus">
  <!-- custom menu or toolbar elements added here -->
</record>
```

The `<record jsxid="customMenus">` element can have either **toolbar** or **menu** child elements. Both the **toolbar** and **menu** elements have three attributes:

- **parent** - The name of the menu or toolbar location. The following values are valid:
 - “MainAppToolbar”
 - “WorkQList”
 - "WorkItemList"
 - “ProcList”
 - “CaseList”
 - “CaseSummary”
- **width** - The display width in pixels.
- **prototype** - The path to the default prototype XML file. This is the TIBCO® TIBCO® General Interface prototype that defines the GUI components for the menu or toolbar.

The custom menu files should be added in a directory under the application root:

```
ClientInstallDir\JSXAPPS\ipc
```

The following example shows a toolbar and a menu element taken from the samples given in `config.xml`:

```
<record jsxid="customMenus">
  <toolbar parent="MainAppToolbar" width="110"
    prototype="JSXAPPS/ipc/custom/prototypes/toolbars/ToolbarSample.xml">
  </toolbar>

  <menu parent="ProcList" width="110"
    prototype="JSXAPPS/ipc/custom/prototypes/menus/MenuSample.xml">
  </menu>
</record>
```

Both the **toolbar** and **menu** elements may also have optional **locale** child elements that define prototype XML files that are localized for specific languages. If a **locale** child element exists, and it corresponds to the language and locale currently selected by the user, the language-specific prototype is loaded, otherwise the default prototype is loaded. For more information on customizing the client application for language localization, see [Localization](#).

Each **locale** child element has three attributes:

- **localeKey** - Locale identifier. This value must correspond to one of the **locale** element **key** attributes defined in the JSXAPPS\ipc\locale\locales.xml configuration file. For example:

```
<locale key="en_US">
</locale>
```

The localeKey attribute values are of the format: ll or ll_CC, where ll is a lowercase, two-letter ISO 639 language code, and CC is the optional, uppercase, two-letter ISO 3166 country code. For a list of codes, visit these web sites:

- International Organization for Standardization (ISO):

<http://www.iso.ch/iso/en/ISOOnline.frontpage>

- Language codes:

<http://www.loc.gov/standards/iso639-2/langhome.html>

- Country codes:

<http://www.iso.ch/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/index.html>

- **width** - The display width in pixels.
- **prototype** - The path to the prototype XML file containing language-specific data. This is the TIBCO® General Interface prototype that defines the GUI components for the menu or toolbar.

The custom localized menu files should be added in a directory under the application root:

ClientInstallDir\JSXAPPS\ipc

The following example shows a toolbar and a menu element, with localized language-specific prototypes, taken from the samples in config.xml:

```
<record jsid="CustomMenu">
  <toolbar parent="MainAppToolbar" width="110"
    prototype="JSXAPPS/ipc/custom/prototypes/toolbars/ToolbarSample.xml">
    <locale localeKey="de_DE" width="200"
      prototype="JSXAPPS/ipc/custom/prototypes/toolbars/ToolbarSample_de_DE.xml"/>
    </toolbar>

  <menu parent="ProcList" width="110"
    prototype="JSXAPPS/ipc/custom/prototypes/menus/MenuSample.xml">
```

There is an example entry in `config.xml` for each of the valid parent attribute values. These examples make reference to the sample files that can be found under the installation home directory:

```
InstallationHomeDir\iprocessclientbrowser\samples\CustomMenus
```

where *InstallationHomeDir* is the directory in which the installer places administrative files, such as the uninstaller, documentation, and sample code. This defaults to `C:\tibco` on Windows systems, and `/opt/tibco` on UNIX systems, but can be specified as a different directory when the iProcess Workspace (Browser) is installed.

The files in the `...\CustomMenus` directory show examples of how to use both menu and toolbar prototype XML files and handle the events using a **CustomEventHandler** class.

The sample custom menus and toolbars can be installed using the steps listed below. These same steps can be used to install actual custom menus and toolbars, substituting the actual custom prototype, JavaScript, and image files and `config.xml` entries. See the TIBCO® General Interface Builder documentation for details on creating custom menu or toolbar prototype files.

1. Create a custom directory under the application root:

```
ClientInstallDir\JSXAPPS\ipc\custom
```

Any valid directory name can be used. The “custom” directory name is used in the sample entries shown in `config.xml`.

2. Copy all of the files from the following directory:

```
InstallationHomeDir\iprocessclientbrowser\samples\CustomMenus
```

... to your custom directory:

```
ClientInstallDir\JSXAPPS\ipc\custom
```

Note that the custom directory structure, and the examples in `config.xml`, include samples for localizing the client application in German (locale `de_DE`). These files provide an example of the structure necessary to provide language-specific support, however, to utilize these localized files, the client application must be configured for German language support (see [Localization](#)).

3. In `config.xml`, add menu or toolbar elements under the **<record jsxid="customMenus">** element. (Uncomment the samples shown in the description for **<record jsxid="customMenus">**.)
4. Add a new mapping record in the `config.xml` file as a child element under **<record jsxid="includes" type="array">** for **CustomEventHandler.js** as shown below:

```
<record jsxid="includes" type="array">
  ...
  <record jsxid="someId" type="map">
    <record jsxid="id" type="string">CustomEventHandler</record>
    <record jsxid="type" type="string">script</record>
    <record jsxid="owner" type="string">application</record>
    <record jsxid="onLoad" type="boolean">true</record>
  </record>
```

Note - The *someId* in the **jsxid** attribute can be any number as long as it's unique among the mapping records in the `config.xml` file.

The application should now load with the sample custom menus and toolbars displayed.

Extending User Access Profiles to Control Custom Menus and Toolbar Buttons

This section describes how to extend the user access profiles to control access to custom menus and toolbar buttons.

The sample code in these instructions, utilizes the sample custom menus and toolbar buttons in the sample files that can be found in the following directory:

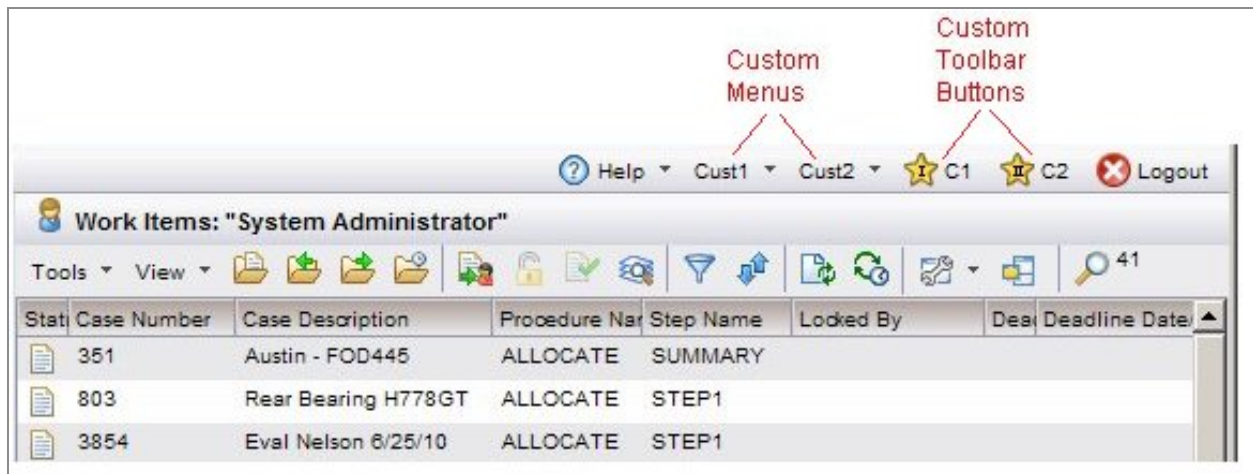
InstallationHomeDir\iproprocessclientbrowser\samples\CustomMenus

Therefore, these instructions assume you have configured custom menus and toolbar buttons as described in [Adding Custom Menu Items and Toolbar Buttons](#) (i.e., you've copied the sample code to a custom directory in the *ClientInstallDir\JSXAPPS\ipc* directory).

In this example, two custom menus and two custom toolbar buttons have been added to the main application toolbar by including the following record in the `JSXAPPS\ipc\config.xml` file:

```
<record jsxid="customMenus">
  <menu parent="MainAppToolbar" width="110"
    prototype="JSXAPPS/ipc/custom/prototypes/menus/MenuSample.xml">
  </menu>
  <toolbar parent="MainAppToolbar" width="110"
    prototype="JSXAPPS/ipc/custom/prototypes/toolbars/ToolbarSample.xml">
```

This causes the following buttons to be displayed on the main application toolbar:



To extend the user access profiles to control these new custom menus and toolbar buttons, follow these steps:

1. Add properties to the user access profiles that control custom menus and toolbar buttons. The user access profiles are defined in the **UserAccessProfiles** record in the following file:

```
ClientInstallDir\JSXAPPS\ipc\userAccessProfiles.xml
```

For more information about user access profiles, see [User Access](#).

The following shows the properties added for our example:

```
<Profile type="Admin" description="Access Level: Admin">
  <!--Optional Type element(s) to assign Profile to other types-->
  <Type name="Admin2" description="Access Level: Admin2"/>
  <Type name="Admin3" description="Access Level: Admin3"/>
  <property name="MainAppToolbar" state="1">
    <property name="bbCustomSample1" state="1"/>
    <property name="bbCustomSample2" state="1"/>
    <property name="mnuCustom1" state="1"/>
    <property name="mnuSample1" state="0"/>
    <property name="mnuSample2" state="1"/>
  </property>
  <property name="mnuCustom2" state="1">
    <property name="mnuSample1" state="0"/>
    <property name="mnuSample2" state="1"/>
  </property>
</Profile>
```

The new properties in this example have been added to the profile for “Admin” users. You will need to add them to all profiles to which you want them to apply (Default, General, etc.).

2. Set the **state** attribute for each new property to the desired state, where “1”= allow access, and “0” = deny access.

The property **state** attribute controls access as follows:

- The “MainAppToolbar” property controls access to the new custom menus and toolbar buttons on the main application toolbar.

- The “tbbCustomSample1” and “tbbCustomSample2” properties control access to the individual custom toolbar buttons.
 - The “mnuCustom1” and “mnuCustom2” properties control access to the individual custom menus.
 - The “mnuSample1” and “mnuSample2” properties control access to the individual selections on the custom menus.
3. Edit your custom event handler class (CustomEventHandler.js in the JSXAPPS\ipc\custom\js directory) to include methods for determining whether the logged in user has access to custom menus and toolbars and take action to either remove or disable items for which the user is not authorized.

An example custom event handler is provided that contains a method named “**authorizeMenus**” for controlling the menus, and a method named “**authorizeToolBar**” for controlling the toolbar buttons. This example event handler is located in the following directory:

InstallHomeDir\iproprocessclientbrowser\samples\CustomMenus\js

4. Make the necessary changes so that the authorization methods are called when toolbars and menus are deserialized. This can be done in one of two ways:
 - The first way is to add the following lines of code to the XML prototypes for the custom menus and toolbars:

To JSXAPPS\ipc\custom\prototypes\menus\MenuSample.xml, add:

```
<onAfterDeserialize><![CDATA[com.xyz.sample.custom.
CustomEventHandler.singleton.authorizeMenus();]]>
</onAfterDeserialize>
```

To JSXAPPS\ipc\custom\prototypes\toolbars\ToolBarSample.xml, add:

```
<onAfterDeserialize><![CDATA[com.xyz.sample.custom.
CustomEventHandler.singleton.authorizeToolBar();]]>
</onAfterDeserialize>
```

- You can also add the method calls to the XML prototypes through GI Builder. To do this, open the prototypes for the custom menus and toolbars in GI Builder, then select ‘Component Profile’. In the Component Profile Editor, enter the following code in the **onAfterDeserialization** field:

For the custom menus prototype, enter:

```
com.xyz.sample.custom.CustomEventHandler.singleton.  
authorizMenus();
```

For the custom toolbars prototype, enter:

```
com.xyz.sample.custom.CustomEventHandler.singleton.  
authorizeToolbars();
```

Your custom menus and toolbar buttons should now react to the access settings in the `userAccessProfiles.xml` file.

Callout Interface



Note

The original callout interface in the TIBCO iProcess Workspace (Browser) was deprecated in version 11.0.0. It was superseded by a simpler method of specifying filters, sorts, and column displays, which is now described in this section. New development should use the callout interface described here.

The deprecated interface is still functional and can continue to be used. The documentation for the original callout interface has been moved to an appendix. See [Deprecated Callout Interface](#).

The callout interface is used to specify filters, sorts, available filter fields, available sort fields, and default column displays for various lists in the client application. The callout interface methods allow you to impose filters and sorts on a work item list or case list when it is initially displayed, or every time it is displayed. You can also force specific columns to be displayed on various lists.



Note

Note that the original callout interface naming convention was to begin each method name with “callout”. It is still called the “callout” interface, although the naming convention for the new methods is to begin each method name with “override”, as they can be used to *override* filter, sort, and column settings in the client application.

Methods in the callout interface can be used in combination with user access profile settings to control filter, sort, and column display. For example, you could use the callout interface methods to set a filter on the case list for a particular user, then use the access profiles to not allow the user to set a filter (i.e., do not give access to the case list **Filter** dialog).

The following bullet items summarize the callout interface methods. Each method is then described in more detail later in this section.

- [overrideFilterFields](#) - Used to modify the filter fields that will appear in the **Field** drop-down list on the **Filter** dialog.
- [overrideInitialFilter](#) - Specifies the *initial* filter for work item and case lists, i.e., it is applied only when the list is initially opened after a login.
- [overrideFilter](#) - Specifies a filter to apply every time a work item or case list is opened.
- [overrideInitialHistoryFilter](#) - Specifies the *initial* filter for case history lists.
- [overrideHistoryFilter](#) - Specifies a filter to apply every time a case history list is displayed, refreshed, or has its filter changed.
- [overrideSortFields](#) - Used to modify the sort fields that will appear in the **Available Fields** list on the **Sort** dialog.
- [overrideInitialSort](#) - Specifies the *initial* sort for work item and case lists, i.e., it is applied only when the list is initially opened after a login.
- [overrideSort](#) - Specifies a sort to apply every time a work item or case list is opened.
- [overrideSelectColumns](#) - Used to modify the columns that appear on the **Column Selector** dialog on various lists.
- [overrideInitialColumns](#) - Used to modify the columns that are displayed when a list is initially loaded.
- [overrideColumns](#) - Used to modify the columns that are displayed when a list is initially loaded, as well as when the columns are modified by the user with the Column Selector.
- [modifyMatrixColumns](#) - Provides direct access to the Matrix control so that specific properties can be changed on the Matrix control and the individual columns.

Sample Callout Handler

The TIBCO iProcess Workspace (Browser) comes with a sample callout handler that contains sample implementations of all of the callout methods.

The sample callout handler is named 'SampleCalloutHandler.js' and is located in the *InstallationHomeDir\iprocessclientbrowser\samples\Callouts* directory, where *InstallationHomeDir* is the directory in which the installer places administrative files, such as the uninstaller, documentation, and sample code. This defaults to C:\tibco on Windows systems, and /opt/tibco on UNIX systems, but can be specified as a different directory when the iProcess Workspace (Browser) is installed.

The following illustrates one of the callout methods in the sample callout handler:

```
ipcClass.prototype.overrideInitialFilter = function(oValue, oContext) {
    jscl.log('overrideInitialFilter called');
    this.logObjectContents(oContext, ' (in) oContext');
    this.logObjectContents(oValue, ' (in) oValue');

    if (oContext.ListType == com.tibco.bpm.ipc.ListContainer.WORKITEM) {
        // Make changes to oValue.Filter for the work items list here

        // SAMPLE CHANGE: append additional filtering information;
        // Append criteria so that the procedure named ALLOCATE is not initially shown.
        // The user will be able to change this later during this session.
        oValue.Filter = this.appendExpression(oContext.Filter, 'SW_PRONAME = "ALLOCATE"');
```

Each of the methods is similar to the method shown above in the following ways:

- All of the callout methods have two parameters: *oValue* and *oContext* (the exception is the **modifyMatrixColumns** method, which has *oMatrix* and *oContext* parameters):
 - *oValue* - This parameter provides the output to the method call. For example, for the **overrideInitialFilter** method shown above, this parameter provides the filter expression needed to modify the initial filter for the work item or case list.
 - *oContext* - This parameter provides all available input to the method. For example, it may provide information about the list upon which you are modifying the filter or sort (user name, list type, etc), or it may provide all of the available fields if you are modifying filter or sort fields.
- Each callout method has a SAMPLE CHANGE example that illustrates making a modification to a filter, sort, filter field, sort, field, or column for a particular list type.
- Each of the callout methods contains if/else statements that provide a location to place code that modifies the *oValue* object for each of the list types for which the method applies.
- You can easily enable or disable the method for each list type by either commenting (to enable) or uncommenting (to disable) the *oValue = null*: line in the

appropriate `if/else` segment for the desired list type:

```
if (oContext.ListType == com.tibco.bpm.ipc.ListContainer.WORKITEM) {
    // Make changes to oValue.Filter for the work items list here

    // SAMPLE CHANGE: append additional filtering information;
    // The initial filter will only show items where the procedure name begins with 'C'.
```

Helper Function

The `SampleCalloutHandler.js` file also contains an **appendExpression** helper function that appends a specified filter expression to the original filter expression. It requires two parameters: *filterExpression*, the original expression, and *appendExpression*, the expression you would like appended to the original.

It is used in the sample code in the **overrideInitialFilter** example shown on [Sample Callout Handler](#).

Configuration

To configure your client application to use the callout handler, perform the following steps:

1. Copy the `SampleCalloutHandler.js` file into a directory you've created under the `ClientInstallDir\JSXAPPS\ipc` directory, where *ClientInstallDir* is the path to the directory in which the client application is installed. For example, `ClientInstallDir\JSXAPPS\ipc\Callouts`.

You may also want to rename the `SampleCalloutHandler.js` file to just `'CalloutHandler.js'`, or to something more specific if that's what its purpose is, for example `'ColumnsCalloutHandler.js'`.

2. Modify the callout handler you copied in step 1 to modify the appropriate lists.
3. Specify the callout handler custom class in the client application's configuration file, `ClientInstallDir\JSXAPPS\ipc\config.xml`.

The **<record jsxid="customCallout">** element specifies which classes will be loaded to handle custom callout methods. The **<Classes>** element can contain any number of **<Class>** elements whose **class** attribute is set to the fully qualified name of the custom class to load. The class is loaded after the user is authenticated at login. This gives the custom class access to the logged-in user's session to query the Action Processor for initialization data, if required.

The following is an example of the `customCallout` element identifying the `ColumnsCalloutHandler` custom class:

```
<record jsid="customCallout" type="ipc">
  <Classes>
    <Class class="com.tibco.bpm.ipc.ColumnsCalloutHandler" />
  </Classes>
</record>
```

4. Add a mapping record to the client application's configuration file, *ClientInstallDir\JSXAPPS\ipc\config.xml* so that it points to your callout handler. Locate the commented-out mapping record as shown below.

```
<!--record jsid="ipc.1" type="map">
  <record jsid="id" type="string">FormTemplate</record>
  <record jsid="type" type="string">script</record>
  <record jsid="owner" type="string">applications</record>
  <record jsid="onLoad" type="boolean">true</record>
  <record jsid="required" type="boolean">true</record>
</record-->
```

Make a copy of the record and remove the comment characters from the copy.

Modify the **id** and **src** records to match the name and location of your callout handler, and change the number in the **jsid** attribute in the first record to any number that is not already used in a mapping record. For example:

```
<record jsid="ipc.2" type="map">
  <record jsid="id" type="string">ColumnsCalloutHandler</record>
  <record jsid="type" type="string">script</record>
  <record jsid="owner" type="string">applications</record>
  <record jsid="onLoad" type="boolean">true</record>
</record>
```

5. Optionally, modify the user access profiles that would be used in conjunction with the custom handling. For example, if your custom handler is setting the default columns on the work item list, you may want to deny access to the Column Selector on the work item list (see **SelectColumns** on [WorkQueue](#) [WorkItem](#) [SelectColumns](#)).



Note

Note that case is significant on some web servers, such as Tomcat. For example, if you are storing your custom callouts in the directory, *ClientInstallDir\JSXAPPS\Callouts* (i.e., with “Callouts” capitalized), the path specification to the custom callout handler in the *config.xml* file cannot be “JSXAPPS/callouts/ColumnsCalloutHandler.js” (i.e., with “callouts” all lowercase).

Callout Methods

The following describes each of the available callout interface methods:

overrideFilterFields

This method allows you to modify the filter fields that will appear in the **Field** drop-down list on the **Filter** dialog. This is used to limit the fields on which the user can filter.

This filter can be applied to work item and case lists — the sample implementation in `SampleCalloutHandler.js` provides `if/else` statements for applying the filter to each type of list.

To remove a field from the **Field** drop-down list, remove it from the array of available fields. Values in the properties of the `AvailableFields` array should not be changed.

This method is called once while initializing the list.

Syntax

```
ipcClass.prototype.overrideFilterFields = function(oValue, oContext)
```

Parameters

- `oValue` - An object that specifies the fields to include in the **Field** drop-down list. It has one property:
 - `oValue.AvailableFields[]` (array) - Describes each available field, as follows:
 - `id` (string) - Identifier for the field.
 - `text` (string) - Text description of the field.
 - `type` (string) - Type of data stored in the field. Possible values are:
 - `swText`
 - `swDate`
 - `swTime`
 - `swNumeric`
 - `swTimeStamp`
 - `swComma`
 - `length` (string) - Maximum length of the field.
 - `regex` (boolean) - True or false, indicating whether a regular expression can be entered.
 - `info` (string) - Text providing information about the format for entering the value. This may be placeholder text that will be replaced by localized text.

`lookup (string)` - Text indicating there was a formatting problem and that a valid example of data follows. This may be placeholder text that will be replaced by localized text.

`validation (string)` - Identifies a method for validating the value entered for comparison. Leave null for text. Possible values are:

```
getDateValidator
getTimeValidator
getTimestampValidator
getNumericValidator
getCommaValidator
getCaseStatusValidator
getZeroOneValidator
```

- `oContext` - An object that provides information about the list being modified. It has the following properties:
 - `oContext.UserName (string)`
 - `oContext.ListType (string)`
 - `oContext.ListTag (string)` - Contains the procedure tag for case lists, and the work queue tag for work item lists.
 - `oContext.ComponentName (string)`
 - `oContext.Filter (string)` - The original filter value.
 - `oContext.AvailableFields[] (Array)` - Contains information about each field that is available. For details, see `oValue.AvailableFields []` above.
 - `oContext.ListName (string)`
 - `oContext.ListDescription (string)`

Returns

The modified `oValue` object, or null if no changes are to be made.

Example

```
ipcClass.prototype.overrideFilterFields = function(oValue, oContext) {
    if (oContext.ListType == com.tibco.bpm.ipc.ListContainer.WORKITEM) {
        // Make changes to oValue.AvailableFields for the work item list here

        // SAMPLE CHANGE: This removes the field SW_QPARAM1 from the available fields list
        oValue.AvailableFields = new Array();
        for (var x=0; x<oContext.AvailableFields.length; x++) {
            if (oContext.AvailableFields[x].id != "SW_QPARAM1") {
                oValue.AvailableFields.push(oContext.AvailableFields[x]);
            }
        }
    }
}
```

overrideInitialFilter

This method specifies the *initial* filter for a list, i.e., it is applied when the list is initially opened after a login and remains in effect until the user removes it or changes it.

An initial filter appears on the **Filter** dialog (and the **Filter** icon has a red check mark), and can be changed by the user if they have access to the **Filter** dialog.

An initial filter can be applied to work item and case lists — the sample implementation in `SampleCalloutHandler.js` provides `if/else` statements for applying the filter to each type of list.

This method is called once upon the initial display of the work item and case list.

Syntax

```
ipcClass.prototype.overrideInitialFilter = function(oValue, oContext)
```

Parameters

- `oValue` - An object that specifies the filter expression to apply to the initial list. It has one property:
 - `oValue.Filter` (string) - The filter expression to apply.
- `oContext` - An object that provides information about the list being modified. It has the following properties:
 - `oContext.UserName` (string)
 - `oContext.ListType` (string)
 - `oContext.ListTag` (string) - Contains the procedure tag for case lists, and the work queue tag for work item lists.

- `oContext.ComponentName` (string)
- `oContext.Filter` (string) - The original filter value.
- `oContext.AvailableFields[]` (Array) - Contains information about each field that is available, as follows:
 - `id` (string) - Identifier for the field.
 - `text` (string) - Text description of the field.
 - `type` (string) - Type of data stored in the field. Possible values are:
 - `swText`
 - `swDate`
 - `swTime`
 - `swNumeric`
 - `swTimeStamp`
 - `swComma`
 - `length` (string) - Maximum length of the field.
 - `regex` (boolean) - True or false, indicating whether a regular expression can be entered.
 - `info` (string) - Text providing information about the format for entering the value. This may be placeholder text that will be replaced by localized text.
 - `lookup` (string) - Text indicating there was a formatting problem and that a valid example of data follows. This may be placeholder text that will be replaced by localized text.
 - `validation` (string) - Identifies a method for validating the value entered for comparison. Leave null for text. Possible values are:
 - `getDateValidator`
 - `getTimeValidator`
 - `getTimeStampValidator`
 - `getNumericValidator`
 - `getCommaValidator`
 - `getCaseStatusValidator`
 - `getZeroOneValidator`
- `oContext.ListName` (string)
- `oContext.ListDescription` (string)
- `oContext.GroupQueue` (boolean) - This is for work item lists only; true or false, indicating whether the work queue is for a queue (true) or a user (false).

Returns

The modified `oValue` object, or null if no changes are to be made.

Example

```
ipcClass.prototype.overrideInitialFilter = function(oValue, oContext) {
    if (oContext.ListType == com.tibco.bpm.ipc.ListContainer.WORKITEM) {
        // Make changes to oValue.Filter for the work items list here

        // SAMPLE CHANGE: append additional filtering information:
        // Append criteria so that the procedure named ALLOCATE is not initially shown.
        // The user will be able to change this later during this session.
        oValue.Filter = this.appendExpression(oContext.Filter, 'SW_PRONAME <> "ALLOCATE"');

        // EXAMPLE: use the GroupQueue context value to determine which ipc toolbox method
        // to call in order to retrieve attributes for either the selected group or user
        // START EXAMPLE - uncomment up to END EXAMPLE to retrieve attributes from the
        // server
        // if (oContext.GroupQueue) {
        //     var attributes = this.app.ipcGetGroupAttributes(oContext.ListName);
        // }
        // else {
        //     var attributes = this.app.ipcGetUserAttributes(oContext.ListName);
        // }
    }
}
```

Note that in the example above, the **appendExpression** helper function is used to append the specified filter expression to the user's specified filter expression. For more information about this helper function, see [Helper Function](#).

overrideFilter

This method is called whenever a work item or case list is opened and each time it is refreshed, allowing modification to the filter that is used.

This filter can be applied to work item and case lists — the sample implementation in `SampleCalloutHandler.js` provides `if/else` statements for applying the filter to each type of list.

The normal use of this callout would be to append additional filtering beyond what the user has specified, but it can be used to make any kind of change to the filter.

The changes to the filter expression applied by this method will NOT appear in the **Filter** dialog. The user will continue to see the original filter there.

Syntax

```
ipcClass.prototype.overrideFilter = function(oValue, oContext)
```

Parameters

- `oValue` - An object that specifies the filter expression to apply to the list. It has one property:
 - `oValue.Filter` (string) - The filter expression to apply.
- `oContext` - An object that provides information about the list being modified. It has the following properties:
 - `oContext.UserName` (string)
 - `oContext.ListType` (string)
 - `oContext.ListTag` (string) - Contains the procedure tag for case lists, and the work queue tag for work item lists.
 - `oContext.ComponentName` (string)
 - `oContext.Filter` (string) - The original filter value.
 - `oContext.AvailableFields[]` (Array) - Contains information about each field that is available to filter on, as follows:
 - `id` (string) - Identifier for the field.
 - `text` (string) - Text description of the field.
 - `type` (string) - Type of data stored in the field. Possible values are:
 - `swText`
 - `swDate`
 - `swTime`
 - `swNumeric`
 - `swTimeStamp`
 - `swComma`
 - `length` (string) - Maximum length of the field.
 - `regex` (boolean) - True or false, indicating whether a regular expression can be entered.
 - `info` (string) - Text providing information about the format for entering the value. This may be placeholder text that will be replaced by localized text.
 - `lookup` (string) - Text indicating there was a formatting problem and that a valid example of data follows. This may be placeholder text that will be replaced by localized text.
 - `validation` (string) - Identifies a method for validating the value entered for comparison. Leave null for text. Possible values are:

```

getDateValidator
getTimeValidator
getTimestampValidator
getNumericValidator
getCommaValidator
getCaseStatusValidator
getZeroOneValidator

```

- `oContext.ListName` (string)
- `oContext.ListDescription` (string)
- `oContext.GroupQueue` (boolean) - This is for work item lists only; true or false, indicating whether the work queue is for a queue (true) or a user (false).

Returns

The modified `oValue` object, or null if no changes are to be made.

Example

```

ipcClass.prototype.overrideFilter = function(oValue, oContext) {
    if (oContext.ListType == com.tibco.bpm.ipc.ListContainer.WORKITEM) {
        // Make changes to oValue.Filter for the work items list here

        // SAMPLE CHANGE: append additional filtering information.
        // Append criteria to the filter so that a procedure named CARPOOL will
        // never be displayed. Note: the user will never see this appended criteria
        // in the filter editor
        oValue.Filter = this.appendExpression(oContext.Filter, SW_PRONAME <> "CARPOOL");

        // EXAMPLE: use the GroupQueue context value to determine which ipc toolbox method
        // to call in order to retrieve attributes for either the selected group or user
        // START EXAMPLE - uncomment up to END EXAMPLE to retrieve attributes from the
        // server
        // if (oContext.GroupQueue) {
        //     var attributes = this.app.ipcGetGroupAttributes(oContext.ListName);
        // }
        // else {
        //     var attributes = this.app.ipcGetUserAttributes(oContext.ListName);
        // }
    }
}

```

Note that in the example above, the **appendExpression** helper function is used to append the specified filter expression to the user's specified filter expression. For more information about this helper function, see [Helper Function](#).

overrideInitialHistoryFilter

This method specifies the *initial* filter for case history lists, i.e., it is applied whenever case history is initially displayed.

This filter is visible and can be changed by the user (unlike a filter applied by the **overrideHistoryFilter** method — see [overrideHistoryFilter](#)).

Syntax

```
ipcClass.prototype.overrideInitialHistoryFilter = function(oValue,
oContext)
```

Parameters

- **oValue** - An object that specifies the filter expression to apply to the initial list. It has one property:
 - **oValue.Filter** (string) - The filter expression to apply.
- **oContext** - An object that provides information about the list being modified. It has the following properties:
 - **oContext.UserName** (string)
 - **oContext.Filter** (string) - The original filter value.

Returns

The modified **oValue** object, or null if no changes are to be made.

Example

```
ipcClass.prototype.overrideInitialHistoryFilter = function(oValue, oContext) {
    // Make changes to oValue.Filter for the case history list here

    // SAMPLE CHANGE: append additional filtering information;
    // Set criteria so that case history entries related to a step named STEPI are
    // always displayed.
    // The user will be able to change this later during this session.
```

overrideHistoryFilter

This method is called upon initial display of the case history list (like the **overrideInitialHistoryFilter** — see [overrideInitialHistoryFilter](#)), plus it is called anytime the user refreshes the list or applies a new filter to the case history list.

The filter applied by this method is *not* visible to the user. It is forcibly applied without the user's knowledge.

Syntax

```
ipcClass.prototype.overrideHistoryFilter = function(oValue, oContext)
```

Parameters

- **oValue** - An object that specifies the filter expression to apply to the list. It has one property:
 - **oValue.Filter** (string) - The filter expression to apply.
- **oContext** - An object that provides information about the list being modified. It has the following properties:
 - **oContext.UserName** (string)
 - **oContext.Filter** (string) - The original filter value.

Returns

The modified **oValue** object, or null if no changes are to be made.

Example

```
ipcClass.prototype.overrideHistoryFilter = function(oValue, oContext) {
    // Make changes to oValue.Filter for the case history list here

    // SAMPLE CHANGE: append additional filtering information:
    // Set criteria so that case history entries related to a step named STEP2 are
    // always displayed.
    oValue.Filter += "STEP_NAME=STEP2";
}
```

overrideSortFields

This method allows you to modify the sort fields that will appear in the **Available Fields** list on the **Sort** dialog. This is used to limit the fields on which the user can sort.

This sort specification can be applied to work item and case lists — the sample implementation in `SampleCalloutHandler.js` provides `if/else` statements for applying the sort to each type of list.

To remove a field from the **Available Fields** drop-down list, remove it from the array of available fields. Values in the properties of the `AvailableFields` array should not be changed.

This method is called once while initializing the list.

Syntax

```
ipcClass.prototype.overrideSortFields = function(oValue, oContext)
```

Parameters

- **oValue** - An object that specifies the fields to include in the **Available Fields** list on the **Sort** dialog. It has one property:
 - **oValue.AvailableFields[]** (Array) - Contains information about each field that is available to sort on, as follows:
 - id** (string) - Identifier for the field.
 - text** (string) - Text description of the field.
 - defaultsorttype** (string) - Default type of data on which this field is sorted. Possible values are:
 - swDateSort
 - swDateTimeSort
 - swNumericSort
 - swTextSort
 - swTimeSort
 - sortas** (boolean) - True or false, indicating whether or not this field can be sorted as a different data type.
- **oContext** - An object that provides information about the list being modified. It has the following properties:
 - **oContext.UserName** (string)
 - **oContext.ListType** (string)
 - **oContext.ListTag** (string) - Contains the procedure tag for case lists, and the work queue tag for work item lists.
 - **oContext.ComponentName** (string)
 - **oContext.AvailableFields[]** (Array) - Contains information about each field that is available to sort on. For details, see **oValue.AvailableFields[]** above.
 - **oContext.ListName** (string)

— `oContext.ListDescription (string)`

Returns

The modified `oValue` object, or null if no changes are to be made.

Example

```
ipcClass.prototype.overrideSortFields = function(oValue, oContext) {
    if (oContext.ListType == com.tibco.bpm.ipc.ListContainer.WORKITEM) {
        // Make changes to oValue.AvailableFields for the work items list here

        // SAMPLE CHANGE: remove SQ_QPARAM1 from list of available fields
        oValue.AvailableFields = new Array();
        for (var x=0; x<oContext.AvailableFields.length; x++) {
            if (oContext.AvailableFields[x].id != "SW_QPARAM1") {
                oValue.AvailableFields.push(oContext.AvailableFields[x]);
            }
        }
    }
}
```

overrideInitialSort

This method specifies the *initial* sort, i.e., it is applied when the list is initially opened after a login and remains in effect until the user removes it or changes it.

This sort specification can be applied to work item and case lists — the sample implementation in `SampleCalloutHandler.js` provides if/else statements for applying the sort to each type of list.

This sort specification appears on the **Sort** dialog after it is applied, and can be changed by the user if they have access to the **Sort** dialog.

This method is called once upon the initial display of the work item and case list.

Syntax

```
ipcClass.prototype.overrideInitialSort = function(oValue, oContext)
```

Parameters

- `oValue` - An object that provides the sort information for the list. It has one property:
 - `oValue.Sort[]` (array) - Each element identifies a field on which the list will be sorted, as follows:

`id` (string) - Identifier for the field used for sorting.

`ascending` (boolean) - True or false, indicating whether the sort order should be ascending (true) or descending (false).

`sorttype` (string) - Type of data on which to sort. Note that the `oContext.AvailableFields.sortas` property identifies whether or not the field can be sorted as a different data type. Possible values are:

```
swDateSort
swDateTimeSort
swNumericSort
swTextSort
swTimeSort
```

- `oContext` - An object that provides information about the list being modified. It has the following properties:

- `oContext.UserName` (string)
- `oContext.ListType` (string)
- `oContext.ListTag` (string) - Contains the procedure tag for case lists, and the work queue tag for work item lists.
- `oContext.ComponentName` (string)
- `oContext.Sort[]` (array) - The original sort information (see `oValue.Sort[]` above).
- `oContext.AvailableFields[]` (Array) - Contains information about each field that is available to sort on, as follows:

`id` (string) - Identifier for the field.

`text` (string) - Text description of the field.

`defaultsorttype` (string) - Default type of data on which this field is sorted. Possible values are:

```
swDateSort
swDateTimeSort
swNumericSort
swTextSort
swTimeSort
```

`sortas` (boolean) - True or false, indicating whether or not this field can be sorted as a different data type.

- `oContext.ListName` (string)

- `oContext.ListDescription` (string)
- `oContext.GroupQueue` (boolean) - This is for work item lists only; true or false, indicating whether the work queue is for a queue (true) or a user (false).

Returns

The modified `oValue` object, or null if no changes are to be made.

Example

```
ipcClass.prototype.overrideInitialSort = function(oValue, oContext) {

  if (oContext.ListType == com.tibco.bpm.ipc.ListContainer.WORKITEM) {
    // Make changes to oValue.Sort for the work items list here

    // SAMPLE CHANGE: force the initial sort to be by case number, descending;
    oValue.Sort = new Array();
    var oSortToAdd = new Object();
    oSortToAdd.id = 'SW_CASENUM';
    oSortToAdd.ascending = false;
    oSortToAdd.sorttype = 'swNumericSort';
    oValue.Sort.push(oSortToAdd);

    // EXAMPLE: use the GroupQueue context value to determine which ipc toolbox method
    // to call in order to retrieve attributes for either the selected group or user
    // START EXAMPLE - uncomment up to END EXAMPLE to retrieve attributes from the
    // server
    // If (oContext.GroupQueue) {
    //   var attributes = this.app.ipcGetGroupAttributes(oContext.ListName);
```

overrideSort

This method is called whenever a work item or case list is opened and each time it is refreshed, allowing modification to the sorting that is used.

This sort specification can be applied to work item and case lists — the sample implementation in `SampleCalloutHandler.js` provides `if/else` statements for applying the sort to each type of list.

The normal use of this callout would be to append additional sort columns to those that the user has specified, but it can be used to make any kind of change to the sorting. The changes made by this method will NOT appear in the **Sort** dialog. The user will continue to see the original sort specification there.

Syntax

```
ipcClass.prototype.overrideSort = function(oValue, oContext)
```

Parameters

- `oValue` - An object that provides the sort information for the list. It has one property:
 - `oValue.Sort[]` (array) - Each element identifies a field on which the list will be sorted, as follows:
 - `id` (string) - Identifier for the field used for sorting.
 - `ascending` (boolean) - True or false, indicating whether the sort order should be ascending (true) or descending (false).
 - `sorttype` (string) - Type of data on which to sort. Note that the `oContext.AvailableFields.sortas` property identifies whether or not the field can be sorted as a different data type. Possible values are:
 - `swDateSort`
 - `swDateTimeSort`
 - `swNumericSort`
 - `swTextSort`
 - `swTimeSort`
- `oContext` - An object that provides information about the list being modified. It has the following properties:
 - `oContext.UserName` (string)
 - `oContext.ListType` (string)
 - `oContext.ListTag` (string) - Contains the procedure tag for case lists, and the work queue tag for work item lists.
 - `oContext.ComponentName` (string)
 - `oContext.Sort[]` (array) - The original sort information (see `oValue.Sort[]` above).
 - `oContext.AvailableFields[]` (Array) - Contains information about each field that is available to sort on, as follows:
 - `id` (string) - Identifier for the field.
 - `text` (string) - Text description of the field.
 - `defaultsorttype` (string) - Default type of data on which this field is sorted. Possible values are:
 - `swDateSort`
 - `swDateTimeSort`
 - `swNumericSort`

swTextSort
swTimeSort

sortas (boolean) - True or false, indicating whether or not this field can be sorted as a different data type.

- oContext.ListName (string)
- oContext.ListDescription (string)
- oContext.GroupQueue (boolean) - This is for work item lists only; true or false, indicating whether the work queue is for a queue (true) or a user (false).

Returns

The modified oValue object, or null if no changes are to be made.

Example

```
ipcClass.prototype.overrideSort = function(oValue, oContext) {
    if (oContext.ListType == com.tibco.bpm.ipc.ListContainer.WORKITEM) {
        // Make changes to oValue.Sort for the work items list here

        // SAMPLE CHANGE: if case number is not already part of the sorting, append
        // it to the end.
        var bFound = false;
        for (var x=0; x<oValue.Sort.length; x++) {
            if (oValue.Sort[x].id == "SW_CASENUM") {
                bFound = true;
            }
        }
        if (!bFound) {
            var oSortToAdd = new Object();
            oSortToAdd.id = "SW_CASENUM";
            oSortToAdd.sorting = false;
            oSortToAdd.sorttype = "swNumericSort";
            oValue.Sort.push(oSortToAdd);
        }

        // EXAMPLE: use the GroupQueue context value to determine which ipc.toolbox method
```

overrideSelectColumns

This method is used to modify the columns that appear on the **Column Selector** dialog. This allows you to specify what columns the user can display on a list through the use of the **Column Selector** dialog.

This can be specified for the following lists:

- work item list
- case list
- work queue list
- procedure list

- outstanding work items list on the case **Outstanding** tab
- outstanding steps to withdraw list on the **Process Jump** dialog

The sample implementation in `SampleCalloutHandler.js` provides `if/else` statements for applying the change to each type of list.

This method is called when the list is initially displayed.

Syntax

```
ipcClass.prototype.overrideSelectColumns = function(oValue, oContext)
```

Parameters

- `oValue` - An object that specifies the columns to display. It has one property:
 - `oValue.Columns[]` (Array) - Contains one element for each column that can be selected, as follows:
 - `id` (string) - Identifies the column.
 - `text` (string)
 - `header` (string)
 - `defaultwidth` (string)
 - `type` (string) - TIBCO® General Interface datatype for the `Matrix.Column` component used to display the data:
 - `Matrix.Column.TYPE_NUMBER` - “number”
 - `Matrix.Column.TYPE_TEXT` (default) [Note: Leave null rather than specifying a value.]
 - `find` (boolean) - True or false indicating whether the column will appear in the find interface.
- `oContext` - An object that provides information about the list being modified. It has the following properties:
 - `oContext.UserName` (string)
 - `oContext.ListType` (string)
 - `oContext.ListTag` (string) - Contains the procedure tag for case lists, the work queue tag for work item lists, and is empty for all other list types.

- `oContext.ComponentName` (string)
- `oContext.AvailableColumns[]` (Array) - Contains information about each column that is available for display:
 - `id` (string) - Identifier for the column.
 - `text` (string)
 - `header` (string)
 - `defaultwidth` (string)
 - `type` (string) - TIBCO® General Interface datatype for the `Matrix.Column` component used to display the data:
 - `Matrix.Column.TYPE_NUMBER` - “number”
 - `Matrix.Column.TYPE_TEXT` (default) [Note: Leave null rather than specifying a value.]
 - `find` (boolean) - True or false indicating whether the column will appear in the find interface.

Plus the following context values appear on work item and case lists:

- `oContext.ListName` (string)
- `oContext.ListDescription` (string)

Plus the following context values appear on the outstanding work item lists on the case **Outstanding** tab and on the **Process Jump** dialog:

- `oContext.CaseTag`
- `oContext.NodeName`
- `oContext.ProcName`
- `oContext.MajorVerion`
- `oContext.MinorVerion`
- `oContext.CaseNumber`

Returns

The modified `oValue` object, or null if no changes are to be made.

Example

```
ipcClass.prototype.overrideInitialColumns = function(oValue, oContext) {
    if (oContext.ListType == com.tibco.bpm.ipc.ListContainer.WORKITEM) {
        // Make changes to oValue.AvailableColumns for the work items list here

        // SAMPLE CHANGE, remove WorkQParam1 from the available columns
        oValue.AvailableColumns = new Array();
        for (var x=0; x<oContext.AvailableColumns.length; x++) {
            if (oContext.AvailableColumns[x].id != "WorkQParam1") {
                oValue.AvailableColumns.push(oContext.AvailableColumns[x]);
            }
        }

        // Note: uncommenting the line below will disable this sample change
        //oValue = null;
    } else if (oContext.ListType == com.tibco.bpm.ipc.ListContainer.CASE) {
```

overrideInitialColumns

This method allows you to modify the columns that are displayed when the following lists are initially loaded:

- work item list
- case list
- work queue list
- procedure list
- outstanding work items list on the case **Outstanding** tab
- outstanding steps to withdraw list on the **Process Jump** dialog

This method is run only upon initial load. This allows you to set the columns in the initial list, then allow the user to modify them with the Column Selector. (Also see the [overrideColumns](#) method; it is run upon initial load, as well as anytime the user changes columns with the Column Selector.)

The sample implementation in `SampleCalloutHandler.js` provides `if/else` statements for applying the change to each type of list.

Syntax

```
ipcClass.prototype.overrideInitialColumns = function(oValue, oContext)
```

Parameters

- `oValue` - An object that specifies the columns to display. It has one property:

- `oValue.Columns[]` (Array) - Contains one element for each column to display, as follows:
 - `id` (string) - Identifies the column.
 - `width` (string) - The width of the column; a default will be used if this is null.
- `oContext` - An object that provides information about the list being modified. It has the following properties:
 - `oContext.UserName` (string)
 - `oContext.ListType` (string)
 - `oContext.ListTag` (string) - Contains the procedure tag for case lists, the work queue tag for work item lists, and is empty for all other list types.
 - `oContext.ComponentName` (string)
 - `oContext.Columns[]` (Array) - Original column information
 - `oContext.AvailableColumns[]` (Array) - Contains information about each column that is available for display, as follows:
 - `id` (string) - Identifier for the column.
 - `text` (string)
 - `header` (string)
 - `defaultwidth` (string)
 - `type` (string) - TIBCO® General Interface datatype for the `Matrix.Column` component used to display the data:
 - `Matrix.Column.TYPE_NUMBER` - “number”
 - `Matrix.Column.TYPE_TEXT` (default) [Note: Leave null rather than specifying a value.]
 - `find` (boolean) - True or false indicating whether the column will appear in the find interface.

Plus the following context values appear on work item and case lists:

- `oContext.ListName` (string)
- `oContext.ListDescription` (string)

Plus the following context values appear on the outstanding work item lists on the case **Outstanding** tab and on the **Process Jump** dialog:

- `oContext.CaseTag`

- oContext.NodeName
- oContext.ProcName
- oContext.MajorVerion
- oContext.MinorVerion
- oContext.CaseNumber

Returns

The modified oValue object, or null if no changes are to be made.

Example

```
ipcClass.prototype.overrideInitialColumns = function(oValue, oContext) {
    if (oContext.ListType == com.tibco.bpm.ipc.ListContainer.WORKITEM) {
        // Make changes to oValue.Columns for the work items list here

        // SAMPLE CHANGE: add the "CaseNumber" column to the list if not already displayed.
        var bFound = false;
        for (var x=0; x<oValue.Columns.length; x++) {
            if (oValue.Columns[x].id == "CaseNumber") {
                bFound = true;
            }
        }
        if (!bFound) {
            oColToAdd = new Object();
            oColToAdd.id = "CaseNumber";
            oColToAdd.width = "100";
            oValue.Columns.push(oColToAdd);
        }
    }
}
```

overrideColumns

This method allows you to modify the columns to display by default. Default columns can be specified for the following lists:

- work item list
- case list
- work queue list
- procedure list
- outstanding work items list on the case **Outstanding** tab
- outstanding steps to withdraw list on the **Process Jump** dialog

The sample implementation in `SampleCalloutHandler.js` provides `if/else` statements for applying the change to each type of list.

This method is called when the list is initially displayed, as well as each time the user changes the columns using the Column Selector. (Also see the [overrideInitialColumns](#) method; it is run only upon initial load of the list.)

Syntax

```
ipcClass.prototype.overrideColumns = function(oValue, oContext)
```

Parameters

- **oValue** - An object that specifies the columns to display. It has one property:
 - **oValue.Columns[]** (Array) - Contains one element for each column to display, as follows:
 - id** (string) - Identifies the column.
 - width** (string) - The width of the column; a default will be used if this is null.
- **oContext** - An object that provides information about the list being modified. It has the following properties:
 - **oContext.UserName** (string)
 - **oContext.ListType** (string)
 - **oContext.ListTag** (string) - Contains the procedure tag for case lists, the work queue tag for work item lists, and is empty for all other list types.
 - **oContext.ComponentName** (string)
 - **oContext.Columns[]** (Array) - Original column information
 - **oContext.AvailableColumns[]** (Array) - Contains information about each column that is available for display, as follows:
 - id** (string) - Identifier for the column.
 - text** (string)
 - header** (string)
 - defaultwidth** (string)
 - type** (string) - TIBCO® General Interface datatype for the `Matrix.Column` component used to display the data:
 - `Matrix.Column.TYPE_NUMBER` - "number"

Matrix.Column.TYPE_TEXT (default) [Note: Leave null rather than specifying a value.]

find (boolean) - True or false indicating whether the column will appear in the find interface.

Plus the following context values appear on work item and case lists:

- oContext.ListName (string)
- oContext.ListDescription (string)

Plus the following context values appear on the outstanding work item lists on the case **Outstanding** tab and on the **Process Jump** dialog:

- oContext.CaseTag
- oContext.NodeName
- oContext.ProcName
- oContext.MajorVersion
- oContext.MinorVersion
- oContext.CaseNumber

Returns

The modified oValue object, or null if no changes are to be made.

Example

```
ipcClass.prototype.overrideColumns = function(oValue, oContext) {
    if (oContext.ListType == com.tibco.bpm.ipc.ListContainer.WORKITEM) {
        // Make changes to oValue.Columns for the work items list here

        // SAMPLE CHNAGE: add the "Proc_Name" column to the list if not already displayed.
        var bFound = false;
        for (var x=0; x<oValue.Columns.length; x++) {
            if (oValue.Columns[x].id == "Proc_Name") {
                bFound = true;
            }
        }
        if (bFound == false) {
            oColToAdd = new Object();
            oColToAdd.id = "Proc_Name";
            oColToAdd.width = "100%";
            oValue.Columns.push(oColToAdd);
        }
    }
}
```

modifyMatrixColumns

This method allows you to modify the **Matrix.Columns** control, which allows you to change specific properties of the columns appearing on the following lists:

- work item list
- case list
- work queue list
- procedure list
- outstanding work items list on the case **Outstanding** tab
- outstanding steps to withdraw list on the **Process Jump** dialog

The sample implementation in `SampleCalloutHandler.js` provides `if/else` statements for applying the change to each type of list.

This method is called when the list is initially displayed, as well as each time the column selection is changed on the list.

Syntax

```
ipcClass.prototype.modifyMatrixColumns = function(oMatrix, oContext)
```

Parameters

- `oMatrix` (`jsx3.gui.Matrix`) - Matrix control that will be used to display the list.
- `oContext` - An object that provides information about the list being modified. It has the following properties:
 - `oContext.UserName` (string)
 - `oContext.ListType` (string)
 - `oContext.ListTag` (string) - Contains the procedure tag for case lists, the work queue tag for work item lists, and is empty for all other list types.
 - `oContext.ComponentName` (string)
 - `oContext.AvailableColumns[]` (Array) - Contains information about each column that is available for display:
 - `id` (string) - Identifier for the column.
 - `text` (string)
 - `header` (string)
 - `defaultwidth` (string)

type (string) - TIBCO® General Interface datatype for the `Matrix.Column` component used to display the data:

`Matrix.Column.TYPE_NUMBER` - "number"

`Matrix.Column.TYPE_TEXT` (default) [Note: Leave null rather than specifying a value.]

`find` (boolean) - True or false indicating whether the column will appear in the find interface.

Plus the following context values appear on work item and case lists:

- `oContext.ListName` (string)
- `oContext.ListDescription` (string)

Plus the following context values appear on the outstanding work item lists on the case **Outstanding** tab and on the **Process Jump** dialog:

- `oContext.CaseTag`
- `oContext.NodeName`
- `oContext.ProcName`
- `oContext.MajorVerion`
- `oContext.MinorVerion`
- `oContext.CaseNumber`

Returns

None.

Example

```
ipcClass.prototype.modifyMatrix.columns = function(oMatrix, oContext) {
    if (oContext.ListType == com.tibco.bpm.ipc.ListContainer.WORKITEM) {
        // Make changes to oMatrix for the work items list here

        // SAMPLE CHANGE: Increases height of rows in the list and changes various
        // properties of the procedure name column

    } else if (oContext.ListType == com.tibco.bpm.ipc.ListContainer.CASE) {
        // Make changes to oMatrix for the case list here
        var cols = oMatrix.getChildren();
        for (var x = 0; x < cols.length; x++) {
            var path = cols[x].getPath();
            if (path == "Proc_Name") {
                cols[x].setCellColor("#FF0000");
                cols[x].setCellFontSize("16");
                cols[x].setCellFontWeight(jsc3.gui.Block.FONTBOLD);
            }
        }
    }
}
```


Browser File Cache Issues

This section describes browser file cache issues that can arise if you modify things such as configuration settings, user access profiles, or custom GI Forms. You need to consider the way these changes are detected by browsers accessing the application.

The following are files that are commonly updated that may cause browser file cache issues:

- the application's configuration file, `config.xml`
- the user access profiles file, `userAccessProfiles.xml`
- the Action Processor's configuration file, `apConfig.xml`
- custom GI Forms in the `\JSXAPPS\ipc\components\Forms` directory
- TIBCO Forms in the WebDAV Server

The web server hosting the iProcess Workspace (Browser) may specify expiration dates for the various files making up the application and its configuration. If the web server does not explicitly set an expiration date, the browser may determine an expiration date based on the creation date and other attributes of the file.

The browser will have a configuration setting that specifies how this expiration date is used. It may choose to continue using a locally cached copy of the file, or it may retrieve a new, possibly updated, copy of the file from the server.

**Note**

Consider how you will need to configure server settings regarding content expiration and what browser settings you want to recommend to users before users begin accessing the application.

Browser File Cache Settings

Microsoft Internet Explorer

Microsoft Internet Explorer lets you choose when to check for newer versions of stored pages. This includes images, media, configuration files, and other application files. You can change this setting as follows:

- **Tools > Internet Options > General** tab. Under **Browsing history**, click on the **Settings** button.

TIBCO recommends one of the first two options: “Every time I visit the webpage” or “Every time I start Internet Explorer”. With these options you will be able to retrieve all configuration and application changes that have been made on the server by either refreshing the page or by exiting and reloading Internet Explorer.

When “Automatically” is selected, content expiration dates, the disk space set aside for temporary internet files, and other internet usage will all affect when updates to the application configuration are used. Some updated files may be retrieved before others, and time periods before new files are retrieved could be minutes, hours, days, or even weeks. However, some web servers may let you specify the expiration dates in a way that will work well with this selection.

The option “Never” will work similar to “Automatically” except that expiration dates are not considered. Locally cached files continue to be used until disk space set aside for temporary files is used up.

Mozilla Firefox

Firefox has similar settings that can be accessed and changed by typing “about:config” into the address bar. The list of configurations displayed include an item entitled “browser.cache.check_doc_frequency”. The values you can specify are:

- 0 - Check once per browser session
- 1 - Check every time the page is viewed
- 2 - Never check (always use cached page)
- 3 (default) - Check when the page is out of date (automatically determined)

You can also configure Firefox so that the cache will be cleared and all browser client files, including updates, will be retrieved the next time the browser client is run by doing the following:

1. On the **Tools** menu, select **Options**.
2. Click on the **Privacy** icon on the top of the dialog.
3. In the **Firefox will** field drop-down list, select “Use custom settings for history”.
4. Check the **Clear history when Firefox closes** check box.
5. Click the **Settings** button.

6. On the **Settings for Clearing History** dialog, check the **Cache** box, then click **OK**.
7. Click **OK** on the **Options** dialog.

How Expiration Dates Are Used

If the web server specifies a content expiration date for a file, and the browser is configured to check for updated files automatically, the date should affect how long a cached version of the file will be used.

If the web server does not specify an expiration date, the browser may determine one based on the file creation date or other attributes.

The details may vary with the browser platform and type/version.

Other factors may also cause a request for a newer file before the expiration date. The local file may be removed from the cache if the disk space set aside for cached files is full, or the user may choose to delete all cached files.

Clearing the Local Browser Cache

In both Microsoft Internet Explorer and Mozilla Firefox, you can delete all locally cached files so that the most recent will be retrieved from the web server. The way in which you do this varies with each browser version; consult your browser's help.

Content Expiration Dates on IIS

If you are hosting the client application or Action Processor on IIS, you can specify expiration dates for the web site, for folders, or for individual files.

Right click on a file or directory inside of the IIS administration application, select the **Properties** option, and switch to the **HTTP Headers** tab. The **Enable Content Expiration** section allows you to configure content expiration.

If the browser is configured to automatically determine when to check for updated files, the expiration date may affect how long the browser uses a locally cached copy of the file before requesting a new, possibly updated, copy.

If you do not enable content expiration dates, the browser may determine an expiration date on its own based on the file creation date or other attributes.

When files are set to immediately expire, the browser should always request a new copy of the file from the web server and never use a locally cached copy. While this should guarantee that updated files are always used, it means the files are transferred with every use. Bandwidth and file transfer speed concerns may not make it the best choice.

If updates to the site are made on a scheduled basis, a date and time can be specified. At that time, updates would be copied to the server and the date/time would be set for the next scheduled update.

Specifying an interval may result in the browser retrieving some updated files before others, on a staggered basis.

Other Considerations and Recommendations

By default, Firefox does not use the file cache for SSL web sites, so if your application or Action Processor is hosted on this type of site, users should always get all updated files.

Content expiration dates cannot be directly configured for static files in Tomcat. However, a custom filter application could be developed to add the content expiration headers to the HTML response that Tomcat creates for the static files. Otherwise, the browser accessing the files will determine an expiration date.

If you are using IIS to host, consider what files you may need to modify in the future. For those files, setting appropriate content expiration may prevent problems with deploying updates.

If setting content expiration dates will not handle problems with deploying updates, recommend appropriate browser settings to users.

Creating New Application Directory for Updates

Another method of deploying changes to a production environment is to create a new application directory where the entire modified application will reside. For example:

- Original: `http://www.myserver.com/IPC_V1/iProcessClient.html`
- Modified: `http://www.myserver.com/ IPC_V2/iProcessClient.html`

Links from other web pages used to launch the client would need to be changed, users notified of the new site if appropriate, and the original URL either redirected or otherwise disabled.

Dynamic Work Item Status Icons Based on Priority

By default, the status icons shown on the work item list is based on the following statuses of the work item:

- Currently locked (open)
- Never been opened (unopened)
- Has been opened (opened)
- Currently suspended (suspended)
- Flagged as urgent (urgent)

This section provides an example of how to display customized status icons based on these same statuses, except instead of using the urgent flag, the priority of each work item is used to determine the status icon to display for the work item.

The following illustrates the icons that are displayed by default for each of the possible combinations of the statuses listed above:

- Locked (Lock.gif) 
- Locked and Urgent (ItemLockedUrgent.gif) 
- Locked and Suspended (ItemLockedSuspended.gif) 
- Locked and Suspended and Urgent (ItemLockedSuspendedUrgent.gif) 
- Unopened (ItemUnopened.gif) 
- Unopened and Urgent (ItemUnopenedUrgent.gif) 
- Unopened and Suspended (ItemUnopenedSuspended.gif) 
- Unopened and Suspended and Urgent (ItemUnopenedSuspendedUrgent.gif) 
- Opened (ItemOpened.gif) 
- Opened and Urgent (ItemOpenedUrgent.gif) 
- Opened and Suspended (ItemOpenedSuspended.gif) 
- Opened and Suspended and Urgent (ItemOpenedSuspendedUrgent.gif) 

This example describes how to modify the XSLT, which transforms the server response containing a list of work items into CDF format for populating a TIBCO® General Interface Matrix component. The changes cause different status icons to be displayed, based on the **Priority** attribute (sso:Priority) of each work item, rather than the **Urgent** attribute (sso:IsUrgent).

To base the icon on priority, rather than urgency, new icons must be created for each identified priority level. In this example, a different icon will be displayed for items with priority ranges 0-59, 60-74, 75-89, and 90 or greater. Work items with a priority less than 60 will display the normal icon. Those in the 60-74, 75-89 and 90 or greater ranges will have different icons instead of the "Urgent" icons shown above.

The following illustrates icons that could be used for each of the priority ranges. You would need to create these icons, with the names shown, to use the example XSLT (or use different names, and modify the example XSLT accordingly):

- Locked and Priority 60-74 (ItemLockedPriority60.gif) 
- Locked and Priority 75-89 (ItemLockedPriority75.gif) 
- Locked and Priority > 90 (ItemLockedPriority90.gif) 
- Locked and Suspended and Priority 60-74 (ItemLockedSuspendedPriority60.gif) 
- Locked and Suspended and Priority 75-89 (ItemLockedSuspendedPriority75.gif) 
- Locked and Suspended and Priority > 90 (ItemLockedSuspendedPriority90.gif) 
- Unopened and Priority 60-74 (ItemUnopenedPriority60.gif) 
- Unopened and Priority 75-89 (ItemUnopenedPriority75.gif) 
- Unopened and Priority > 90 (ItemUnopenedPriority90.gif) 
- Unopened and Suspended and Priority 60-74 (ItemUnopenedSuspendedPriority60.gif) 
- Unopened and Suspended and Priority 75-89 (ItemUnopenedSuspendedPriority75.gif) 
- Unopened and Suspended and Priority > 90 (ItemUnopenedSuspendedPriority90.gif) 
- Opened and Priority 60-74 (ItemOpenedPriority60.gif) 
- Opened and Priority 75-89 (ItemOpenedPriority75.gif) 

- Opened and Priority > 90 (ItemOpenedPriority90.gif) 
- Opened and Suspended and Priority 60-74 (ItemOpenedSuspendedPriority60.gif) 
- Opened and Suspended and Priority 75-89 (ItemOpenedSuspendedPriority75.gif) 
- Opened and Suspended and Priority > 90 (ItemOpenedSuspendedPriority90.gif) 

To modify the XSLT to set the status image based on priority values:

1. Save the new icons that are based on priority (e.g., ItemLockedPriority60.gif) to the following directory, with the file names shown above:

JSXAPSS\ipc\application\images

2. Make a backup copy of the following file:

JSXAPPS\ipc\components\ListContainer\xsl\WorkItem\actionProcessorToCdf.xsl

3. Open the following file with an editor:

JSXAPPS\ipc\components\ListContainer\xsl\WorkItem\actionProcessorToCdf.xsl

4. Search for the following template in the actionProcessorToCdf.xsl file:

```
<xsl:template name="setStatusImage" >
```

5. Replace the “setStatusImage” template with the following template definition. This template is modified to replace each test for the Urgent attribute with three tests for the Priority attribute in the specified ranges:

```
<xsl:template name="setStatusImage">
  <!-- This template defines the rules for outputting the image
  status icon -->
  <xsl:choose>
    <xsl:when test="sso:IsLocked = 'true' or sso:IsLongLocked =
    'true'">
      <xsl:choose>
        <xsl:when test="sso:Priority > 59 and sso:Priority <=
        75">
          <xsl:attribute name="IsStatus">true</xsl:attribute>
        <xsl:choose>
          <xsl:when test="sso:IsSuspended = 'true'">
            <xsl:attribute
            name="IsStatusImage">JSXAPPS/ipc/application/images/ItemLockedSuspe
            ndedPriority60.gif</xsl:attribute>
          </xsl:when>
          <xsl:otherwise>
            <xsl:attribute
```

```

name="IsStatusImage">JSXAPPS/ipc/application/images/ItemLockedPrior
ity60.gif</xsl:attribute>
    </xsl:otherwise>
  </xsl:choose>
</xsl:when>
<xsl:when test="sso:Priority > 74 and sso:Priority <
90">
    <xsl:attribute name="IsStatus">true</xsl:attribute>
    <xsl:choose>
      <xsl:when test="sso:IsSuspended = 'true'">
        <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemLockedSuspe
ndedPriority75.gif</xsl:attribute>
      </xsl:when>
      <xsl:otherwise>
        <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemLockedPrior
ity75.gif</xsl:attribute>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:when>
  <xsl:when test="sso:Priority > 89">
    <xsl:attribute name="IsStatus">true</xsl:attribute>
    <xsl:choose>
      <xsl:when test="sso:IsSuspended = 'true'">
        <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemLockedSuspe
ndedPriority90.gif</xsl:attribute>
      </xsl:when>
      <xsl:otherwise>
        <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemLockedPrior
ity90.gif</xsl:attribute>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:when>
  <xsl:otherwise>
    <xsl:attribute name="IsStatus">true</xsl:attribute>
    <xsl:choose>
      <xsl:when test="sso:IsSuspended = 'true'">
        <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemLockedSuspe
nded.gif</xsl:attribute>
      </xsl:when>
      <xsl:otherwise>
        <xsl:attribute

```



```

name="IsStatusImage">JSXAPPS/ipc/application/images/Lock.gif</xsl:attribute>
    </xsl:otherwise>
  </xsl:choose>
</xsl:otherwise>
</xsl:choose>
</xsl:when>
<xsl:when test="sso:IsUnopened = 'true' and sso:IsSuspended = 'true'">
  <xsl:choose>
    <xsl:when test="sso:Priority > 59 and sso:Priority < 75">
      <xsl:attribute name="IsStatus">true</xsl:attribute>
      <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemUnopenedSuspendedPriority60.gif</xsl:attribute>
    </xsl:when>
    <xsl:when test="sso:Priority > 74 and sso:Priority < 90">
      <xsl:attribute name="IsStatus">true</xsl:attribute>
      <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemUnopenedSuspendedPriority75.gif</xsl:attribute>
    </xsl:when>
    <xsl:when test="sso:Priority > 89">
      <xsl:attribute name="IsStatus">true</xsl:attribute>
      <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemUnopenedSuspendedPriority90.gif</xsl:attribute>
    </xsl:when>
    <xsl:otherwise>
      <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemUnopenedSuspended.gif</xsl:attribute>
      <xsl:attribute name="IsStatus">true</xsl:attribute>
    </xsl:otherwise>
  </xsl:choose>
</xsl:when>
<xsl:when test="sso:IsUnopened = 'true'">
  <xsl:choose>
    <xsl:when test="sso:Priority > 59 and sso:Priority < 75">
      <xsl:attribute name="IsStatus">true</xsl:attribute>
      <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemUnopenedPriority60.gif</xsl:attribute>
    </xsl:when>
    <xsl:when test="sso:Priority > 74 and sso:Priority <

```

```

90">
    <xsl:attribute name="IsStatus">true</xsl:attribute>
    <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemUnopenedPri
ority75.gif</xsl:attribute>
  </xsl:when>
  <xsl:when test="sso:Priority > 89">
    <xsl:attribute name="IsStatus">true</xsl:attribute>
    <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemUnopenedPri
ority90.gif</xsl:attribute>
  </xsl:when>
  <xsl:otherwise>
    <xsl:attribute name="IsStatus">true</xsl:attribute>
    <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemUnopened.gi
f</xsl:attribute>
  </xsl:otherwise>
</xsl:choose>
</xsl:when>
<xsl:when test="sso:IsUnopened = 'false' and sso:IsSuspended =
'true'">
  <xsl:choose>
    <xsl:when test="sso:Priority > 59 and sso:Priority <
75">
      <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemOpenedSuspe
ndedPriority60.gif</xsl:attribute>
      <xsl:attribute name="IsStatus">true</xsl:attribute>
    </xsl:when>
    <xsl:when test="sso:Priority > 74 and sso:Priority <
90">
      <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemOpenedSuspe
ndedPriority75.gif</xsl:attribute>
      <xsl:attribute name="IsStatus">true</xsl:attribute>
    </xsl:when>
    <xsl:when test="sso:Priority > 89">
      <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemOpenedSuspe
ndedPriority90.gif</xsl:attribute>
      <xsl:attribute name="IsStatus">true</xsl:attribute>
    </xsl:when>
    <xsl:otherwise>
      <xsl:attribute name="IsStatus">true</xsl:attribute>
      <xsl:attribute

```

```

name="IsStatusImage">JSXAPPS/ipc/application/images/ItemOpenedSuspe
nded.gif</xsl:attribute>
    </xsl:otherwise>
</xsl:choose>
</xsl:when>
<xsl:when test="sso:IsUnopened = 'false'">
    <xsl:choose>
        <xsl:when test="sso:Priority > 59 and sso:Priority <
75">
            <xsl:attribute name="IsStatus">true</xsl:attribute>
            <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemOpenedPrior
ity60.gif</xsl:attribute>
            </xsl:when>
            <xsl:when test="sso:Priority > 74 and sso:Priority <
90">
                <xsl:attribute name="IsStatus">true</xsl:attribute>
                <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemOpenedPrior
ity75.gif</xsl:attribute>
                </xsl:when>
                <xsl:when test="sso:Priority > 89">
                    <xsl:attribute name="IsStatus">true</xsl:attribute>
                    <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemOpenedPrior
ity90.gif</xsl:attribute>
                    </xsl:when>
                    <xsl:otherwise>
                        <xsl:attribute name="IsStatus">true</xsl:attribute>
                        <xsl:attribute
name="IsStatusImage">JSXAPPS/ipc/application/images/ItemOpened.gif<
/xsl:attribute>
                        </xsl:otherwise>
                    </xsl:choose>
                </xsl:when>
            </xsl:choose>
        </xsl:template>

```

The resulting work item list, containing all icon combinations, displays as follows:

1. Make a backup copy of the following file:

JSXAPPS\ipc\components\ListContainer\prototypes\WorkItem\listDefault.xml

2. Open the following file with an editor:

JSXAPPS\ipc\components\ListContainer\prototypes\WorkItem\listDefault.xml

3. Remove the following XML element from the listDefault.xml file and save the file:

```
<xsl:parameters jsx_rowbg1="#ffffeff" jsx_rowbg2="#e8e7e9"/>
```

Modify the XSLT to set the background color of rows based on CDQP and step name values:

1. Make a backup copy of the following file:

JSXAPPS\ipc\components\ListContainer\xsl\WorkItem\actionProcessorToCdf.xsl

2. Open the following file with an editor:

JSXAPPS\ipc\components\ListContainer\xsl\WorkItem\actionProcessorToCdf.xsl

3. Locate the following template in the actionProcessorToCdf.xsl file:

```
<xsl:template match="sso:vCDQP" >
```

4. Replace this template with the following template definition. This template adds a test for the value of the "OCCUPATION" CDQP, and based on whether the value is "Engineer", "Lawyer" or "Physician", performs a subsequent test on the value of the StepName attribute. Each row of the list will have a different color, depending on the CDQP value and the step name of the work item.

```
<xsl:template match="sso:vCDQP" >
  <xsl:variable name="cdqpName" select="sso:FieldName" />
  <xsl:attribute name="{ $cdqpName }">
    <xsl:value-of select="sso:Value" />
  </xsl:attribute>
  <xsl:choose>
    <xsl:when test="$cdqpName = 'OCCUPATION'">
      <xsl:choose>
        <xsl:when test="sso:Value = 'Engineer'">
          <xsl:choose>
            <xsl:when test="ancestor::sso:vWorkItem/sso:StepName =
'APPL'">
              <xsl:attribute name="jsxstyle">background-
color:#fff6d2;</xsl:attribute>
            </xsl:when>
            <xsl:when test="ancestor::sso:vWorkItem/sso:StepName =
'APPROVAL'">
              <xsl:attribute name="jsxstyle">background-
```

```

color:#deedd4;</xsl:attribute>
    </xsl:when>
    <xsl:when test="ancestor::sso:vWorkItem/sso:StepName =
'NOTIFY'">
        <xsl:attribute name="jsxstyle">background-
color:#d0a3a3;</xsl:attribute>
        </xsl:when>
    </xsl:choose>
</xsl:when>
<xsl:when test="sso:Value = 'Lawyer'">
    <xsl:choose>
        <xsl:when test="ancestor::sso:vWorkItem/sso:StepName =
'APPL'">
            <xsl:attribute name="jsxstyle">background-
color:#f5d972;</xsl:attribute>
            </xsl:when>
            <xsl:when test="ancestor::sso:vWorkItem/sso:StepName =
'APPROVAL'">
                <xsl:attribute name="jsxstyle">background-
color:#a0b493;</xsl:attribute>
                </xsl:when>
                <xsl:when test="ancestor::sso:vWorkItem/sso:StepName =
'NOTIFY'">
                    <xsl:attribute name="jsxstyle">background-
color:#e47474;</xsl:attribute>
                    </xsl:when>
                </xsl:choose>
            </xsl:when>
            <xsl:when test="sso:Value = 'Physician'">
                <xsl:choose>
                    <xsl:when test="ancestor::sso:vWorkItem/sso:StepName =
'APPL'">
                        <xsl:attribute name="jsxstyle">background-
color:#debb5c;</xsl:attribute>
                        </xsl:when>
                        <xsl:when test="ancestor::sso:vWorkItem/sso:StepName =
'APPROVAL'">
                            <xsl:attribute name="jsxstyle">background-
color:#82b75f;</xsl:attribute>
                            </xsl:when>
                            <xsl:when test="ancestor::sso:vWorkItem/sso:StepName =
'NOTIFY'">
                                <xsl:attribute name="jsxstyle">background-
color:#d84545;</xsl:attribute>
                                </xsl:when>
                            </xsl:choose>
                        </xsl:when>
                    </xsl:choose>
                </xsl:when>
            </xsl:choose>
        </xsl:when>
    </xsl:choose>

```

```

</xsl:when>
</xsl:choose>
</xsl:template>

```

The resulting work item list appears as:

Work Items: swadmin						
Stat	Case Number	Case Description	Occupation	Step Name	Procedure Na	Locked By
	20129	case 1	Engineer	APPROVAL	ORDER	
	20179	case 12	Lawyer	APPROVAL	ORDER	
	20179	case 12	Lawyer	NOTIFY	ORDER	
	20229	case 13	Engineer	APPL	ORDER	
	20180	case 14	Physician	APPL	ORDER	
	19980	case 15	Lawyer	APPROVAL	ORDER	
	19980	case 15	Lawyer	NOTIFY	ORDER	
	19881	case 16	Engineer	APPL	ORDER	
	20135	case 17	Physician	APPL	ORDER	
	20230	case 18	Lawyer	APPL	ORDER	
	20231	case 19	Engineer	APPROVAL	ORDER	
	20231	case 19	Engineer	NOTIFY	ORDER	
	20031	case 2	Engineer	APPROVAL	ORDER	
	20133	case 9	Physician	NOTIFY	ORDER	
	20133	case 9	Physician	APPROVAL	ORDER	

Configuring the Action Processor

The installation procedure for the TIBCO iProcess Workspace (Browser) steps you through the configurations that are necessary to get the Action Processor up and running.

This section describes additional options that can be configured after the Action Processor has been installed.

Overview

All of the Action Processor configuration settings are specified in the Action Processor's configuration file, which is located as follows:

```
APDir\apConfig.xml
```

where *APDir* is the directory in which the Action Processor is installed (which defaults to "TIBCOActProc"). The location of this directory on your system will depend on which Web Application Server is hosting your Action Processor.

If changes are made to any of the Action Processor configuration settings, you must restart the Action Processor so it picks up the changes to *apConfig.xml*. To restart the Action Processor, stop and restart the Web Application Server.

The Action Processor configuration settings are described in the following subsections.



Note

Note that Secure Sockets Layer (SSL) can be used to secure communications between the Action Processor and the client. However, the way in which it is implemented is specific to the Web Application Server (WAS) you are using. See the documentation for your WAS.

Also note that if you access the TIBCO iProcess Workspace (Browser) through HTTPS, you must also access the Action Processor through HTTPS, and vice versa. In other words, you cannot access one through HTTP and the other one through HTTPS.

Log Settings

There are a number of configuration parameters that allow you to specify the type and amount of information to write to the Action Processor log file. Locate the following elements in the `apConfig.xml` file and change the default value to fit your needs for Action Processor logging:

- **<LogLevel>** - This specifies the level of information to write to the log. The valid levels are:
 - **ERROR** - This provides the least amount of information.
 - **WARN** - This provides more information than ERROR, but less than INFO.
 - **INFO** - (The default) This provides more information than WARN, but less than DEBUG.
 - **DEBUG** - This provides the most amount of information. This setting should be used only if there is a problem that is being diagnosed.
- **<MaxLogFileSize>** - This specifies the maximum size of the log file before it is rolled over. The default is 10 MB. Setting this to 0 (zero) causes the log to never roll over.
- **<LogArchiveCount>** - This specifies the number of archived log files. These are created if the log exceeds the maximum size limit specified with the **<MaxLogFileSize>** tag. The default is 5. Note that the naming convention for these differs between Java and .NET, as follows (assuming you are using the default log file name specified in the **<LogFile>** tag):
 - **Java Action Processor: `APLog.log.X`**, where X starts at 1 and increases as archive log files are created (e.g., **`APLog.log.1`**).
 - **.NET Action Processor: `APLogXXX.log`**, where XXX starts at 001 and increases as archive log files are created (e.g., **`APLog001.log`**).
- **<LogFile>** - This specifies the fully qualified path to the log file. If only the name is specified, the log file is written to the directory specified in the system property **`user.home`**. This defaults to **`APLog.log`**.

XML Response Compression

This parameter specifies whether or not to compress the XML response from the Action Processor. (This is only applicable for the Java Action Processor.) This may be desired if

the connection link between the client and Action Processor is slow.

The default is to not compress the response.

To specify the XML response compression setting:

1. Locate the **<IsCompressResponse>** element in the `apConfig.xml` file:

```
<IsCompressResponse>false</IsCompressResponse>
```

2. Specify a value in this element of “true” to compress the XML response from the Action Processor, or “false” to not compress.

Return Request Parameters

This configuration setting specifies whether or not the Action Processor should, by default, return parameter information to the client (parameter information is returned to the client in the form of **<InParam>** elements).

When the Action Processor calls the **GetXMLResult** method, it passes the value in this configuration setting in the “**WithInput**” parameter to specify whether or not to return input information. This setting defaults to “**false**”.

To specify the return request parameter setting:

1. Locate the **<IsReturnSSOParams>** element in the `apConfig.xml` file:

```
<IsReturnSSOParams>false</IsReturnSSOParams>
```

2. Specify a value in this element of “true” to include input parameters by default, or “false” to exclude input parameters by default.

External Form URI

This parameter is used to specify a URI that points to an external forms package that is used to display forms in the iProcess Workspace (Browser). This parameter is used with the following external forms packages:

- **TIBCO BusinessWorks™ iProcess Forms Plug-in** - This plug-in allows you to create a form using TIBCO BusinessWorks™ FormBuilder, then associate that form with a

step in a TIBCO iProcess Engine procedure. For information about this plug-in, see the *TIBCO BusinessWorks iProcess Forms Plug-in User's Guide*.

- **ASP Forms** - These forms can be created in a .NET project, then used in a TIBCO iProcess Engine procedure. For information about using these types of forms, see [ASP Forms](#).
- **JSP Forms** - These forms can be created in a Java development environment, then used in a TIBCO iProcess Engine procedure. For information about using these types of forms, see [JSP Forms](#).

To specify the external form URI setting:

1. Locate the **<ExternalFormURI>** element in the `apConfig.xml` file:

```
<ExternalFormURI>http://localhost:8080/</ExternalFormURI>
```

2. Replace “localhost” with the name of the computer on which the Formflow process, ASP form, or JSP form is deployed, and replace “8080” with the port number used by the Web Application Server that is hosting your external form. For example:

```
<ExternalFormURI>http://whisler:90/</ExternalFormURI>
```

Note that the final slash following the port number is required.

Obfuscating External Form URI Information

When an external form is invoked, the user name, password, and server detail information can either be displayed in the URI, or they can be obfuscated, i.e., made obscure by showing asterisks in their place.

The default is “false”, which causes the user name, password, and server details to be displayed in the URI.

To change the obfuscation setting:

1. Locate the desired obfuscation element in the `apConfig.xml` file:

```
- <IsObfuscateExternalFormURIUsername>
- <IsObfuscateExternalFormURIPassword>
- <IsObfuscateExternalFormURIServerDetails>
```

2. Set the desired element to “true” to obfuscate the information, or “false” to display the information.

Server Factories

This parameter specifies the server factories to use when using the Remote Method Invocation (RMI). (This parameter is only applicable for the Java Action Processor.)

The TIBCO iProcess Workspace (Browser) installer asks for server factory information if the interface type is RMI. It then writes that information into the **<ServerFactories>** element in the `apConfig.xml` file.

You can modify the server factory information, or add additional **<ServerFactory>** elements if multiple server factories are used.

To configure the server factories to use:

1. Locate the **<ServerFactories>** element in the `apConfig.xml` file:

```
<ServerFactories>
  <ServerFactory>
    <UniqueId>prServerFactory/UniqueId</UniqueId>
    <Name>prServerFactory</Name>
    <IsJRMP>true</IsJRMP>
    <LoadFactor>100</LoadFactor>
  </ServerFactory>
</ServerFactories>
```

2. Enter values in these elements to specify information about the server factory:

- **<UniqueId>**: Used to uniquely identify the server factory.
- **<Name>**: The name of the server factory.
- **<IsJRMP>**: This specifies the protocol used to marshal objects. Setting this to “true” causes the Java Remote Method Protocol (JRMP) to be used; setting this to “false” causes Internet Inter-ORB Protocol (IIOP) to be used. (Note that the protocol specified here must be the same protocol used when constructing **xSession** objects.)
- **<LoadFactor>**: This specifies how the load should be dispersed among the server factories when using multiple server factories. The number specified here is viewed as a percentage of the total of the load factors specified for all server factories. For ease of use, the total of all load factors should total 100 (although it’s not required). For example, if three server factories are used, their load factors might be set to 50, 30, and 20 — the first one will get 50% of the load, the second 30%, and the third 20%.

This defaults to 100. You can modify this value if you use multiple server factories.

- **<JNDIEnv>**: Multiple JNDI environment settings can be specified to use when creating the context used to locate the server factory. Each of these will contain a **<Name>** and **<Value>** element to provide specifics about the environment setting.

The installer asks for a “Java naming provider URL” (which specifies the location of the registry when the registry is being used as the initial context). The installer adds this value to one set of JNDI environment **<Name>** / **<Value>** element pairs (see the example above).

Another set of **<Name>** / **<Value>** element pairs is provided that specifies the initial context factory to use (see `java.naming.factory.initial` in the example above).

XML Validation

This parameter specifies whether or not to validate the XML requests to the Action Processor using an XSD (**apAction.xsd**). This is configurable because there is some overhead incurred when validating requests.

The default is to not validate the requests.

To specify the XML validation setting:

1. Locate the **<IsValidateActionXML>** element in the `apConfig.xml` file:

```
<IsValidateActionXML>false</IsValidateActionXML>
```

2. Specify a value in this element of “true” to validate the XML requests, or “false” to not validate.

CSRF Validation

This is a new configuration setting introduced in version 11.8, as part of the security feature enhancement. It specifies whether to validate the CSRF requests and defines if the validation is for FORMS requests.

By default, the element is defined to validate all the requests, including Forms requests. So the default value is 1.

To specify the CSRF validation requests:

Locate the `<AcroSticLevel>` element:

```
<AcroSticLevel>1</AcroSticLevel>
```

Specify one of the following values in this element:

0 to signify there is no CSRF validation.

1 to signify the validation is for all the requests, including FORMS requests.

2 to signify the validation is for all the requests, excluding FORMS Requests.



Note

To enable additional security, ensure that the AcroStic Level value is not zero and set xId in the Login ID tag. For details on how to set xId in the Login ID tag, refer to the **Authentication** section of the **TIBCO iProcess Workspace (Browser) Action Processor Reference**.

Action Processor Version

This parameter is used to specify whether or not to display the Action Processor version number in the Action Processor action response. The version number is displayed under the `<ap:Status>` element in the response: For example:

```
<ap:Status>
  <ap:Version>11.3.0</ap:Version>
  ...
</ap:Status>
```

To specify this setting:

1. Locate the `<IsReturnVersion>` element in the `apConfig.xml` file:

```
<IsReturnVersion>true</IsReturnVersion>
```

2. Specify a value in this element of “true” to return the Action Processor version, or “false” to not return the version.

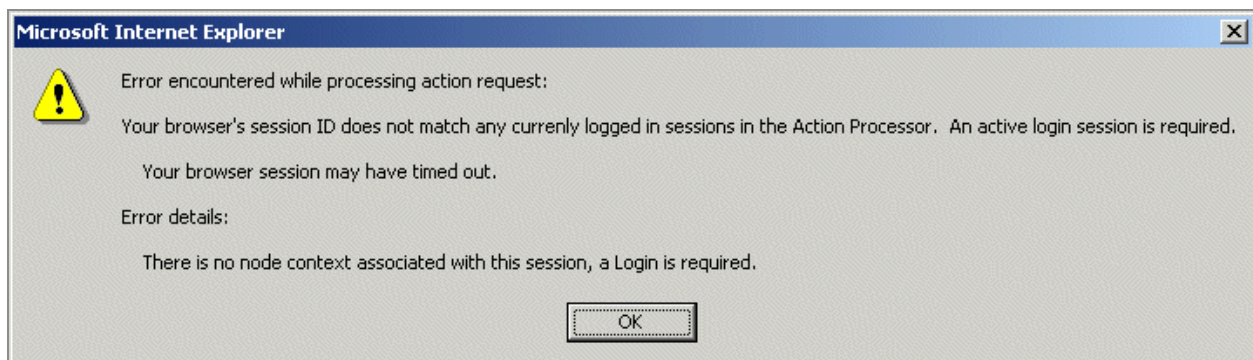
Application Server Settings

This section describes some configuration settings in Web Application Servers (WAS) that can be useful when configuring your TIBCO iProcess Workspace (Browser) system.

For a comprehensive discussion of WAS configuration settings, see the documentation for your WAS.

Session Timeout

The session timeout specifies the number of minutes in which the communication session between the WAS and the client will timeout if there is no user activity, which results in the Action Processor timing out. If the session times out, a message similar to the following is displayed:



The following subsections describe how to change the session timeout in Tomcat, WebLogic, and IIS. If you are using a different application server, see the documentation for that server.



Note

Also see the [Session Monitor](#) parameter on [Session Timeout](#).

Tomcat Session Timeout

By default, the session timeout on Tomcat is 30 minutes. To specify a different session timeout:

1. Add the **<session-timeout>** element to the following configuration file:

```
TomcatDir\webapps\APDir\WEB-INF\web.xml
```

where *TomcatDir* is the directory in which Tomcat is installed, and *APDir* is the directory in which the Action Processor is installed (which defaults to **TIBCOActProc**).

The **<session-timeout>** element must be added to the **<session-config>** element, for example:

```
<session-config>
  <cookie-config>
    <name>
      ACROSTIC_ID
    </name>
  </cookie-config>
  <session-timeout>30</session-timeout>
</session-config>
```

2. Set the value of the **<session-timeout>** element to the number of minutes for the timeout. A value of -1 causes the session to never timeout.

You must restart the Action Processor for the change to take effect.

WebLogic Session Timeout

By default, the session timeout on WebLogic is 60 minutes. To specify a different session timeout:

3. Add the **<session-timeout>** element in the following configuration file:

```
WebLogicDir\webapps\APDir\WEB-INF\web.xml
```

where *WebLogicDir* is the directory in which WebLogic is installed, and *APDir* is the directory in which the Action Processor is installed (which defaults to **TIBCOActProc**).

The **<session-timeout>** element must be added to the **<session-config>** element, for example:

```
<session-config>
  <cookie-config>
    <name>
      ACROSTIC_ID
    </name>
  </cookie-config>
  <session-timeout>30</session-timeout>
</session-config>
```

4. Set the value of the **<session-timeout>** element to the number of minutes for the timeout. A value of -1 causes the session to never timeout.

You must restart the Action Processor for the change to take effect.

IIS Session Timeout

By default, the session timeout on IIS is 20 minutes. To specify a different session timeout:

5. Locate the **<sessionState>** element in the following configuration file:

```
\Inetpub\wwwroot\APDir\Web.config
```

where *APDir* is the directory in which the Action Processor is installed (which defaults to **TIBCOActProc**).

For example:

```
<sessionState
  mode="InProc"
  stateConnectionString="tcpip=127.0.0.1:42424"
  sqlConne&#0;tionString="data          source=127.0.0.1;Trusted_
Connection=yes"
  cookieless="false"
  timeout="20"
/>
```

6. Set the value of the **timeout** attribute for the desired number of minutes for the session timeout.

You must restart the Action Processor for the change to take effect.

Maximum POST Size

This section is applicable only to users of Tomcat.

By default, Tomcat sets a limit of 2097152 (2 MB) on the maximum size for HTTP POST requests it will accept. This is specified in the **maxPostSize** attribute in the following configuration file:

```
TomcatDir\conf\server.xml
```

where *TomcatDir* is the directory in which Tomcat is installed.

It is possible to exceed the maximum POST size limit in situations where your application contains large memos or lots of fields.

Any large POST request to the Action Processor that exceeds the limit might result in one of the following errors:

```
java.lang.IllegalStateException: Post too large
```

—or—

```
Error encountered while processing action request:
```

```
There was no Action specified.
```

```
Error Details:
```

```
No additional details available. Check the Action Processor log.
```

For additional information, see the Apache Tomcat documentation:

<http://tomcat.apache.org/>

Character Encoding

The following describes how to set character encoding to UTF-8 using Tomcat. If you are using a different web server, refer to their documentation.

If you are going to be using any double-byte character encoding, such as the Japanese character sets, you need to set the **URIEncoding** attribute in Tomcat's `server.xml` file to **"UTF-8"**, otherwise data such as the case description will not be properly encoded. The **URIEncoding** attribute must be located in the **<Connector>** element.

Note that the **URIEncoding** parameter is not in the `server.xml` file by default; you must add it if you are going to be using any double-byte character sets.

To do this:

1. Locate the **<Connector>** element in the following configuration file:

```
TomcatDir\conf\server.xml
```

where *TomcatDir* is the directory in which Tomcat is installed.

2. Add a **URIEncoding** attribute and set it to “UTF-8”. See the following example:

```
<Connector URIEncoding="UTF-8" port="8080"
maxHttpHeaderSize="8192" maxThreads="150" minSpareThreads="25"
maxSpareThreads="75" enableLookups="false" redirectPort="8443"
acceptCount="100" connectionTimeout="20000"
disableUploadTimeout="true" maxPostSize="0" />
```

You must restart Tomcat for the change to take effect.

Java Heap Size

This section is applicable only to users of Tomcat.

If you are running a Java Action Processor in Tomcat, and receive a “Java heap space” error message when attempting to display a list that contains a large number of items (10000+), you will need to do the following:

Modify the **EXECJAVA** statement in the **catalina.bat** file (Windows) or **catalina.sh** file (UNIX) to include the **-Xms** and **-Xmx** parameters. For example:

```
%_EXECJAVA% %JAVA_OPTS% %CATALINA_OPTS% %DEBUG_OPTS% -Xms1024m -
Xmx1024m -Djava.library.path="C:\Java\Tomcat55\shared\lib" -
Djava.endorsed.dirs="%JAVA_ENDORSED_DIRS%" -classpath "%CLASSPATH%" -
Dcatalina.base="%CATALINA_BASE%" -Dcatalina.home="%CATALINA_HOME%" -
Djava.io.tmpdir="%CATALINA_TMPDIR%" %MAINCLASS% %CMD_LINE_ARGS% %ACTION%
```

Direct Login

This section describes methods of bypassing the iProcess Workspace (Browser) **Login** screen by passing login credentials directly.

Direct Login

You can bypass the iProcess Workspace (Browser) **Login** screen by passing login credentials directly into the application.



Users can also be authenticated using credentials they have already entered in another web application by using the *single authentication* feature. For information, see [Single Authentication](#).

Login credentials can be directly passed into the iProcess Workspace (Browser) in one of the following ways:

- On the URL
- In an HTML form element named 'DirectLogin'
- In an HTML script element that defines 'getDirectLoginArgs'

When loading, the application will look for direct login credentials in the order shown above.

To use any of the methods of direct login listed above, direct login must be enabled in the iProcess Workspace (Browser) configuration file. This is described in [Enabling Direct Login](#).

In all of the methods of direct login, the following case-insensitive parameters can be specified:

- **Username** - The user name of the user logging in.
- **Password** - The password of the user logging in.
- **ComputerName** - The computer name for the iProcess Objects Server.

- **IPAddress** - The IP address of the machine on which the TIBCO iProcess Objects Server is installed. You can specify the name of the host machine, as long as that name resolves to the IP address of the machine where the iProcess Objects Server is running. Note, however, that this name must be able to be resolved by the machine on which the Action Processor is running.
- **TCPPort** - The TCP Port for the iProcess Objects Server.
- **Name** - The iProcess Engine node name.
- **Director** - “true” or “false”, indicating if an iProcess Objects Director is being used.
- **persistOnServer** - (optional) If “true”, user preferences are persisted on, and read from, the server. If “false”, user preferences are stored locally, as well as read from the local machine upon login.

Default = “false” if attribute is absent

- **maxDataSize** - (optional) This is the maximum number of characters in a property value string. This must be set at or below the field size supported by the database used on the server. The maximum number is typically 256K. For double-byte character encoding, the maximum value is 128K.

Default = 32768 (32K) if attribute is absent

Minimum value = 10

- **ServerName** - If specified, this causes the connection information to be obtained from the **ServerNodes** record in the client’s `config.xml` file. Pass the node name in this parameter, i.e., the name in the **<Name />** element. For example, if the following server node is specified in the client’s `config.xml` file, and you want to connect to that node via direct login, pass “phoenix” in this parameter:

```
<record jsid="ServerNodes" type="ipc">
  <record displayName="The Phoenix Server">
    <NodeId>
      <ComputerName>EastServer</ComputerName>
      <IPAddress>10.67.2.50</IPAddress>
      <TCPPort>5897</TCPPort>
      <Name>phoenix</Name>
    
```

- **BaseUrl** - The Action Processor base URL. If this is passed in the URL, it overrides the baseUrl specified in the client’s `config.xml` file. If it is not passed, the baseUrl specified in the client’s `config.xml` file is used.

The following sections provide examples of each of the three direct login methods.

Enabling Direct Login

To be able to use any of the methods of direct login, direct login must be enabled in the iProcess Workspace (Browser).

To enable direct login:

1. Open the *ClientInstallDir\JSXAPPS\ipc\config.xml* file with an editor.
2. Locate the **Login** record in the *config.xml* file:

```
<record jsxid="Login" type="ipc" useRemember="true"
allowDirectLogin="false"/>
```

3. Modify the **allowDirectLogin** attribute as follows:
 - “true” enables the use of the methods of direct login listed on [Direct Login](#).
 - “false” disables direct login.

On the URL

The following are forms and examples of passing login credentials on the URL:

- Providing all needed information in the URL:

```
http://ClientHost:Port/ClientDir/iProcessClient.html?
Username=xxx&Password=xxx&ComputerName=xxx&IPAddress=xxx&
tcpport=xxx&name=xxx&director=xxx&persistOnServer=xxx&
maxDataSize=xxx
```

For example:

```
http://mymachine:7979/TIBCOiPCInt/iProcessClient.html?username=swad
min&password=&computername=liberty&ipaddress=10.20.30.40&tcpport=58
221&name=v11&director=false&persistOnServer=true&maxDataSize=2222
```

- Using the **ServerName** parameter, which specifies a node name. It will use the connection information for that node in the **<ServerNodes/>** element in the client's *config.xml* file:

```
http://ClientHost:Port/ClientDir/iProcessClient.html?
Username=xxx&Password=xxx&ServerName=xxx
```

For example:

```
http://mymachine:7979/TIBCOiPCInt/iProcessClient.html?username=swadmin&password=&servername=v11
```

- Using the **baseUrl** parameter, which overrides the baseUrl specified in the client's config.xml file:

```
http://ClientHost:Port/ClientDir/iProcessClient.html?
Username=xxx&Password=xxx&ServerName=xxx&BaseUrl=xxx
```

For example:

```
http://mymachine:7979/TIBCOiPCInt/iProcessClient.html?username=swadmin&password=&servername=v11&BaseUrl=http%3A%2F%2Fozquadling%3A8090%2Fap1120rc2%2FActionProcessor.servlet
```

If the **BaseUrl** parameter is passed on the URL, it must be URI encoded as shown in the example above.



Note

Note that case is not significant for parameter/argument names when using direct login.

In an HTML Form Element Named 'DirectLogin'

The following is an example HTML form:

```
<form name="DirectLogin">
  <input type="hidden" name="BaseUrl"
    value="http://ServerComputerName:90/ipc/ActionProcessor.aspx">
  <input type="hidden" name="Username" value="swadmin">
  <input type="hidden" name="Password" value="">
  <!--If ServerName is specified, this will override other
    values and will be looked up from the config.xml file. -->
  <!-- <input type="hidden" name="ServerName"
    value="MyServerConfigName"> -->
  <input type="hidden" name="ComputerName"
    value="ServerComputerName">
  <input type="hidden" name="IPAddress" value="10.20.30.40">
  <input type="hidden" name="TCPPort" value="54321">
```

```

<input type="hidden" name="Name" value="ServerNodeName">
<input type="hidden" name="Director" value="false">
<input type="hidden" name="persistOnServer" value="true">
<input type="hidden" name="maxDataSize" value="32768">
</form>

```

Add this form to the HTML file used to launch the iProcess Workspace (Browser), which by default is *ClientInstallDir/iProcessClient.html*, where *ClientInstallDir* is the path to the directory in which the iProcess Workspace (Browser) is installed.

In an HTML Script Element that Defines ‘getDirectLoginArgs’

The following is an example HTML script:

```

<script language="JavaScript">
    function getDirectLoginArgs(nameSpace) {
        var args = new Object();
        if (nameSpace == 'wccApp') {
            args.BaseUrl = "http://ServerComputerName:90/" +
                           "ipc/ActionProcessor.aspx";
            args.Username = "swadmin";
            args.Password = "";
            //If ServerName is specified, this will override other
            //values and will be looked up from the config.xml file.
            //args.ServerName = "MyServerConfigName";
            args.ComputerName = "ServerComputerName";
            args.IPAddress = "10.20.30.40";
            args.TCPPort = "54321";
            args.Name = "ServerNodeName";
            args.Director = "false";
            args.persistOnServer = "true";
            args.maxDataSize = "32768";
        }
        return args;
    }
</script>

```

Add this script to the HTML file used to launch the iProcess Workspace (Browser), which by default is *ClientInstallDir/iProcessClient.html*, where *ClientInstallDir* is the path to the directory in which the iProcess Workspace (Browser) is installed.

Single Authentication

This section describes how to configure the TIBCO iProcess Workspace (Browser) so that users can be authenticated using credentials they have already entered in another web application.

Introduction

The *single authentication* feature allows users to go directly to the iProcess Workspace (Browser) from another web application without going through the iProcess Workspace (Browser) **Login** screen. When configured for this feature, the system will authenticate the user using the credentials the user has already entered in the other web application.

The single authentication feature provides a secure method of passing the user's credentials to the iProcess Workspace (Browser).

The way in which you implement this feature depends on whether you are using the Java or .NET Action Processor. Each is described in the following sections.

Single authentication is supported in the client application, as well as in custom applications created with WCC components.



Note

Other methods of passing login credentials to perform a “direct login” are described in [Direct Login](#).



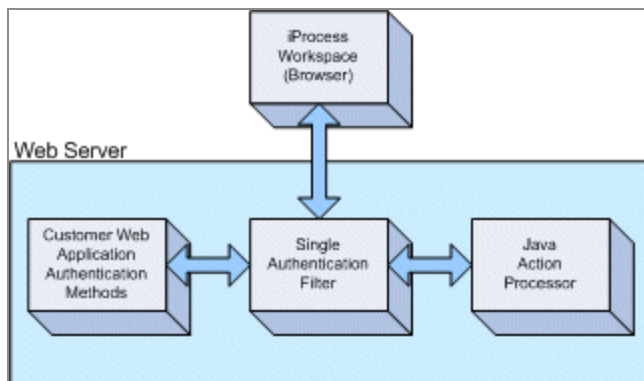
Note

Also note that there is a configuration parameter (**logoutUrl**) that allows you to redirect the client upon a user logout. This feature can also be used with single authentication. For more information, see [Redirecting Client to URL on Logout](#).

Java Single Authentication

The Java single authentication feature is used if you are using the Java Action Processor. It allows users to go directly to the iProcess Workspace (Browser) from another web application in which they have already been authenticated.

The Java single authentication feature implements a *filter* to act as a mediator between the customer's web application authentication methods and the iProcess Workspace (Browser) and Java Action Processor.



Web Server Configuration

The single authentication filter is configured by adding the **<filter>** and **<filter-mapping>** elements to the web server's deployment descriptor file, which is located as follows:

```
APInstallDir\WEB-INF\web.xml
```

where *APInstallDir* is the installation directory of the Java Action Processor, which defaults to **TIBCOActProc**.

The following shows the elements that must be added to **web.xml**:

```

<filter>
  <filter-name>LoginAuthenticator</filter-name>
  <filter-class>com.tibco.bpm.ap.filter.AuthenticateFilter</filter-class>
  <init-param>
    <param-name>AuthenticatorClass</param-name>
    <param-value>
      com.tibco.bpm.ap.filter.sample.SampleAuthenticator
    </param-value>
  </init-param>

```

```

</filter>
<filter-mapping>
  <filter-name>LoginAuthenticator</filter-name>
  <servlet-name>ActionProcessor</servlet-name>
</filter-mapping>

```

The **<init-param>** element provides initialization parameters, including the class name of the authenticator plug-in that the customer must implement. See the next section for more information about this plug-in.

Authenticator Plug-in

The customer must create a plug-in that implements the **Authenticator** interface, as follows:

```

public interface Authenticator {
    public String getUsername();
    public String getPassword();
    public String getNodeName();
    public String getComputerName();
    public String getIpAddress();
    public int getTcpPort();
    public boolean isDirector();
    public boolean authenticate (HttpServletRequest);
}

```

The **authenticate** method receives the request object so it can sync its authentication parameters with the request ID. The **authenticate** method's return value is boolean, indicating either success or failure of the authenticate call.

The **Authenticator** interface also provides getter methods for the information required for a successful login to the iProcess Workspace (Browser). The filter uses these values to construct a **Login** action request to the Java Action Processor.

iProcess Workspace (Browser) Configuration

To use single authentication, you must enable it in the iProcess Workspace (Browser) configuration file, which is located as follows:

```
ClientInstallDir\JSXAPPS\ipc\config.xml
```

Locate the **SingleAuthentication** record in the configuration file, and set **useSingle** to true:

```
<record jsxid="SingleAuthentication" type="ipc" useSingle="true"
failureUrl="" />
```

This informs the iProcess Workspace (Browser) that an external application will be providing the login authentication credentials.

The **failureUrl** attribute is used to specify a URL to which the user is redirected if the login fails.

- If there is a value specified in the **failureUrl** attribute, and the login fails, an alert dialog is displayed containing the message "User authentication could not be confirmed. You will be redirected to an appropriate login page." The browser is then redirected to the specified URL.
- If the value of **failureUrl** is empty, and the login fails, an alert dialog is displayed containing the message "User authentication could not be confirmed. You must successfully log in before using this site."

Java Single Authentication Sample

A sample Java single authentication plug-in is included with the TIBCO iProcess Workspace (Browser). This section describes how to set up the sample plug-in, as well as information about how it operates.

The Java single authentication sample is provided in the following two JAR files:

- **SingleAuthenticationSample.jar** - This JAR file contains the files that make up the sample authenticator. It includes:
 - login.html
 - customApplication.html
 - WEB-INF/web.xml
 - WEB-INF/classes/com/tibco/bpm/ap/filter/sample/Login.class
 - WEB-INF/classes/com/tibco/bpm/ap/filter/sample/SampleAuthenticator.class

- `src/com/tibco/bpm/ap/filter/sample/Login.java`
- `src/com/tibco/bpm/ap/filter/sample/SampleAuthenticator.java`
- **SingleAuthentication.jar** - This JAR file contains the class files used by the authenticator plug-in. It includes:
 - `com/tibco/bpm/ap/filter/Authenticator.class`
 - `com/tibco/bpm/ap/filter/AuthenticateFilter.class`
 - `com/tibco/bpm/ap/filter/AuthenticateRequestWrapper.class`
 - `com/tibco/bpm/ap/filter/xml/Attribute.class`
 - `com/tibco/bpm/ap/filter/xml/Element.class`
 - `com/tibco/bpm/ap/filter/xml/action/Action.class`
 - `com/tibco/bpm/ap/filter/xml/request/Login.class`
 - `com/tibco/bpm/ap/filter/xml/request/Request.class`
 - `com/tibco/bpm/ap/filter/xml/request/UserId.class`
 - `com/tibco/bpm/ap/filter/xml/vobject.NodeId.class`

These JAR files are located in the following directory:

```
InstallationHomeDir\iprocessclientbrowser\samples\
SingleAuthentication\Java
```

where *InstallationHomeDir* is the directory in which the installer places administrative files, such as the uninstaller, documentation, and sample code. This defaults to `C:\tibco` on Windows systems, and `/opt/tibco` on UNIX systems, but can be specified as a different directory when the iProcess Workspace (Browser) is installed.

Setting Up the Java Single Authentication Sample

To set up the Java single authentication sample, follow these steps:

1. Stop your web application server.
2. Unpack the `SingleAuthenticationSample.jar` file into a temporary directory.
3. Compare the `APInstallDir/WEB-INF/web.xml` file to the one supplied in the `SingleAuthenticationSample.jar` file. The one provided in the

SingleAuthenticationSample.jar file will contain the following new sections: <welcome-file-list>, <filter>, <filter-mapping>. It will also contain additional <servlet> and <servlet-mapping> sections for the **Login.do** servlet. Assuming these are the only differences, you can safely replace the *APInstallDir*/WEB-INF/web.xml file with the one supplied in the SingleAuthenticationSample.jar file.

4. Create the following directory:

```
APInstallDir/WEB-INF/classes/com/tibco/bpm/ap/filter/sample
```

5. From the SingleAuthenticationSample.jar file you unpacked in [Unpack the SingleAuthenticationSample.jar file into a temporary directory.](#), copy the Login.class and SampleAuthenticator.class files to the directory you created in [Create the following directory.](#)
6. From the SingleAuthenticationSample.jar file you unpacked in [Unpack the SingleAuthenticationSample.jar file into a temporary directory.](#), copy the login.html and customerApplication.html files to the Action Processor installation directory.
7. Copy the SingleAuthentication.jar file to the *APInstallDir*/WEB-INF/lib directory.
8. Configure the iProcess Workspace (Browser) to use single authentication — see [iProcess Workspace \(Browser\) Configuration](#).
9. Restart your web application server.

Launching the Java Single Authentication Sample

To launch the Java single authentication sample, follow these steps:

1. Execute the following URL in your browser:

```
http://Host:Port/APDir/login.html
```

where:

- *Host* is the name of the machine hosting the Action Processor.
- *Port* is the port number used by the WAS that is hosting the Action Processor to communicate with web applications.
- *APDir* is the directory on *Host* in which the Action Processor is installed. This defaults to **TIBCOActProc**.

This presents a dialog on which you can enter login credentials. This represents the outside source from which login credentials are provided.

2. Enter a user name and password, as well as information about the iProcess Objects Server to log into, then click the **Submit** button.

This causes the login credentials to be sent to the **login.do** servlet, which stores the information in the session using a Java hash map. It then launches the `customerApplication.html` file, which represents the customer application. This page contains the following fields: **New window URL**, **Window width**, and **Window height**.

3. Enter the URL to the iProcess Workspace (Browser) in the **New window URL** field. You can also specify new width and height values, if desired.

The URL must be in the form:

```
http://Host:Port/iProcessClientDir/iProcessClient.html
```

where:

- *Host* is the name of the machine hosting the iProcess Workspace (Browser).
 - *Port* is the port number used by the WAS that is hosting the iProcess Workspace (Browser) to communicate with web applications.
 - *iProcessClientDir* is the directory on *Host* in which the iProcess Workspace (Browser) is installed. This defaults to **TIBCOiPCInt**.
4. Click the **Open new window** button.

Since the iProcess Workspace (Browser) has been configured to use the single authentication feature (see [Configure the iProcess Workspace \(Browser\) to use single authentication — see iProcess Workspace \(Browser\) Configuration](#). on [Configure the iProcess Workspace \(Browser\) to use single authentication — see iProcess Workspace \(Browser\) Configuration](#).), it will bypass the normal iProcess Workspace (Browser) **Login** screen and send a login request to the Action Processor.

The filter (**AuthenticateFilter**), which sits between the iProcess Workspace (Browser) and the Action Processor (see the illustration on [The Java single authentication feature implements a filter to act as a mediator between the customer's web application authentication methods and the iProcess Workspace \(Browser\) and Java Action Processor](#).), wraps each requests in an **AuthenticateRequestWrapper** object. The wrapper examines each request for a single-authenticate login request. When it finds one, it uses the **SampleAuthenticator**, which implements the **Authenticator**

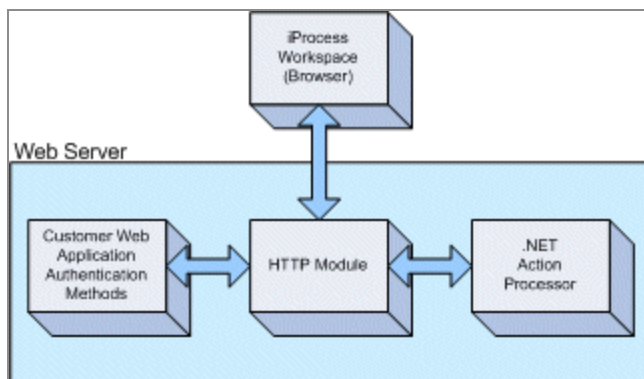
interface, calling the **authenticate** method, and passing the request object. The **authenticate** method obtains the login data hash map from the session, and queries the hash map to populate its login properties. The **AuthenticateRequestWrapper** then uses the **SampleAuthenticator** properties to construct a login request that is passed to the Action Processor.

If the previously entered credentials are valid, the iProcess Workspace (Browser) is displayed.

.NET Single Authentication

The .NET single authentication feature is used if you are using the .NET Action Processor. It allows users to go directly to the iProcess Workspace (Browser) from another web application in which they have already been authenticated.

The .NET single authentication feature implements an *HTTP module* that is inserted into the HTTP request pipeline to act as a mediator between the customer's web application authentication methods and the iProcess Workspace (Browser) and .NET Action Processor.



Web Server Configuration

The HTTP module is hooked into the pipeline by configuring it in the web server configuration file, which is located as follows:

```
C:\inetpub\wwwroot\APInstallDir\web.config
```

where *APInstallDir* is the installation directory of the .NET Action Processor, which defaults to **TIBCOActProc**.

The following shows the elements that must be added to `web.config`:

```
<configuration>
  <appSettings>
    <add key="AuthenticatorAssemblyPath"
        value="D:\\SingleAuthenticationSample.dll" />
    <add key="AuthenticatorAssemblyName"
        value="SingleAuthenticationSample" />
    <add key="AuthenticatorAssemblyImpl"
        value="SingleAuthenticationSample.SampleAuthenticator" />
    <add key="AuthenticatorLogFile"
        value="C:\\temp\\SingleAuthentication.log" />
  </appSettings>
  <system.web>
    <httpModules>
      <add name="LoginAuthenticator"
          type="SingleAuthentication.AuthenticateFilter,
SingleAuthentication" />
    </httpModules>
  </system.web>
</configuration>
```

where:

- **AuthenticatorAssemblyPath** is the absolute path to the customer assembly.
- **AuthenticatorAssemblyName** is the name of the assembly.
- **AuthenticatorAssemblyImpl** is the name of the `IAuthenticator` implementation.
- **AuthenticatorLogFile** is the absolute path to the log file.

Authenticator Plug-in

The customer must create a plug-in that implements the **SingleAuthentication.IAuthenticator** interface, as follows:

```
using Sysyem;
using System.Web;
namespace SingleAuthentication {
  public interface IAuthenticator {
    string UserName { get; }
    string Password { get; }
    string NodeName { get; }
    string ComputerName { get; }
  }
}
```

```

    string IPAddress { get; }
    int TcpPort { get; }
    bool Director { get; }
    bool Authenticate (HttpContext ctx);
}
}

```

The customer plug-in **Authenticator** module implements an **Authenticate** method that receives the **HttpContext** object so that it can sync its authentication parameters with the context. The return value of the **Authenticate** method is a boolean, indicating success or failure of the authenticate call.

The **IAuthenticator** interface also provides getter methods for the information required for a successful login to the iProcess Workspace (Browser). The HTTP module uses these values to construct a **Login** action request to the .NET Action Processor.

iProcess Workspace (Browser) Configuration

To use single authentication, you must enable it in the iProcess Workspace (Browser) configuration file, which is located as follows:

```
ClientInstallDir\JSXAPPS\ipc\config.xml
```

Locate the **SingleAuthentication** record in the configuration file, and set **useSingle** to true:

```

<record jsxid="SingleAuthentication" type="ipc" useSingle="true"
failureUrl="" />

```

This informs the iProcess Workspace (Browser) that an external application will be providing the login authentication credentials.



Note

The **useSingle** parameter can also be passed in the URL, if desired. The following example URL is telling the client that an outside source is supplying the login credentials:

```
http://somedirectory/iProcessClient.html?useSingle=true
```

The **failureUrl** attribute is used to specify a URL to which the user is redirected if the login fails.

- If there is a value specified in the **failureUrl** attribute, and the login fails, an alert dialog is displayed containing the message “User authentication could not be confirmed. You will be redirected to an appropriate login page.” The browser is then redirected to the specified URL.
- If the value of **failureUrl** is empty, and the login fails, an alert dialog is displayed showing the message “User authentication could not be confirmed. You must successfully log in before using this site.”

.NET Single Authentication Sample

A sample .NET single authentication plug-in is included with the TIBCO iProcess Workspace (Browser). This section describes how to set up the sample plug-in, as well as information about how it operates.

The sample .NET single authentication plug-in defines three pages:

- SingleAuthenticationLoginSample.aspx
- customerApplication.htm
- iProcessClient.htm

The SingleAuthenticationLoginSample.aspx page provides the user the ability to define login particulars. Once the data has been filled in, the user will select the **Login** button. This action sends login data to the **event** method, which stores it in the session using a C# hash table. The **event** method then launches customerApplication.htm.

The customerApplication.htm page provides the user the ability to specify the URL from which to launch the iProcess Workspace (Browser) application, as well as the window width and window height. The customerApplication.htm page also contains an **Open new window** button, which launches the application specified in the URL field in a new browser window using the specified width and height settings.

Since the iProcess Workspace (Browser) application has been configured for single authentication, it bypasses the client's login window, and sends a single authentication request to the Action Processor. The **AuthenticateFilter** is inserted into the Action Processor request pipeline, and it examines each request looking for the single authentication request. When it finds a single authentication request, it uses the sample object, **SampleAuthenticator**, which implements the **IAuthenticator** interface, calling the

Authenticate method and passing the **HttpContext** object. The **Authenticate** method obtains the login data hash table from the session and queries the hash table to populate it with the login properties. The **AuthenticateFilter** then uses the **SampleAuthenticator** to construct a login request that is passed to the Action Processor.

**Note**

This sample assumes that `SingleAuthenticationLoginSample.aspx`, `customerApplication.htm`, `iProcessClient.htm`, and the TIBCO iProcess Workspace (Browser) application itself will reside in the Action Processor installation directory. (The iProcess Workspace (Browser) application must reside in the Action Processor installation directory for this sample because of .NET security constraints.)

Setting Up the .NET Single Authentication Sample

The .NET single authentication sample plug-in includes the following **.zip** file and assembly:

- `SingleAuthenticationSample.zip`
- `SingleAuthentication.dll`

These files are located in the following directory:

`InstallationHomeDir\iprocessclientbrowser\samples\SingleAuthentication\dotNet`

where *InstallationHomeDir* is the directory in which the installer places administrative files, such as the uninstaller, documentation, and sample code. This defaults to `C:\tibco` on Windows systems, and `/opt/tibco` on UNIX systems, but can be specified as a different directory when the iProcess Workspace (Browser) is installed.

To set up the .NET single authentication sample, following these steps:

1. Unpack the `SingleAuthenticationSample.zip` file into a temporary location.

This zip file contains all of the files needed for the plug-in sample (including the sample source code), which include:

- `AssemblyInfo.cs`
- `SingleAuthenticationLoginSample.cs`
- `SampleAuthenticator.cs`

- bin\Release\SingleAuthenticationSample.dll
 - SingleAuthenticationLoginSample.aspx
 - SingleAuthenticationLoginSample.aspx.resx
 - Web.config
 - customerApplication.htm
 - iProcessClient.htm
2. Stop Microsoft IIS.
 3. Compare the *APIInstallDir\Web.config* file to the one supplied in the *SingleAuthenticationSample.zip* file. They should differ by the <appSettings> and <httpModules> sections. Assuming these are the only differences, you can safely replace the *APIInstallDir\Web.config* file with the one supplied in the *SingleAuthenticationSample.zip* file.
 4. Copy the assemblies bin\Release\SingleAuthenticationSample.dll and SingleAuthentication.dll into the *APIInstallDir\bin* directory.
 5. In the new *APIInstallDir\Web.config* file, modify the **key=AuthenticatorAssemblyPath** value in the <appSettings> section to point to the absolute location where you installed the SingleAuthenticationSample.dll file (*APIInstallDir\bin*).
 6. Copy the files SingleAuthenticationLoginSample.aspx, SingleAuthenticationLoginSample.aspx.resx, customerApplication.htm and iProcessClient.htm to the Action Processor installation directory.
 7. Install a copy of the TIBCO iProcess Workspace (Browser) in the Action Processor installation directory. This is required for this sample because of .NET security constraints.
 8. Configure the iProcess Workspace (Browser) to perform a single authentication. For information about how to do this, see [iProcess Workspace \(Browser\) Configuration](#).
 9. Restart Microsoft IIS.

Launching the .NET Single Authentication Sample

To launch the .NET single authentication sample, follow these steps:

1. Execute the following URL in your browser:

```
http://Host:Port/APDir/SingleAuthenticationLoginSample.aspx
```

where:

- *Host* is the name of the machine hosting the Action Processor.
- *Port* is the port number used by IIS to communicate with web applications.
- *APDir* is the directory on *Host* in which the Action Processor is installed. This defaults to **TIBCOActProc**.

This presents a dialog on which you can enter login credentials. This represents the outside source from which login credentials are provided.

2. Enter a user name and password, as well as the information about the iProcess Objects Server to log into, then click the **Submit** button.

The `customerApplication.htm` page is opened. This page contains the following fields: **New window URL**, **Window width**, and **Window height**. The **New window URL** field will be prefilled with “iProcessClient.htm”. You can specify different width and height values, if desired.

3. Click the **Open new window** button.

If the login particulars you entered in step 2 are correct, the iProcess Workspace (Browser) will start without displaying its **Login** screen.

Logging

This section describes the logs available in TIBCO iProcess Workspace (Browser).

Introduction

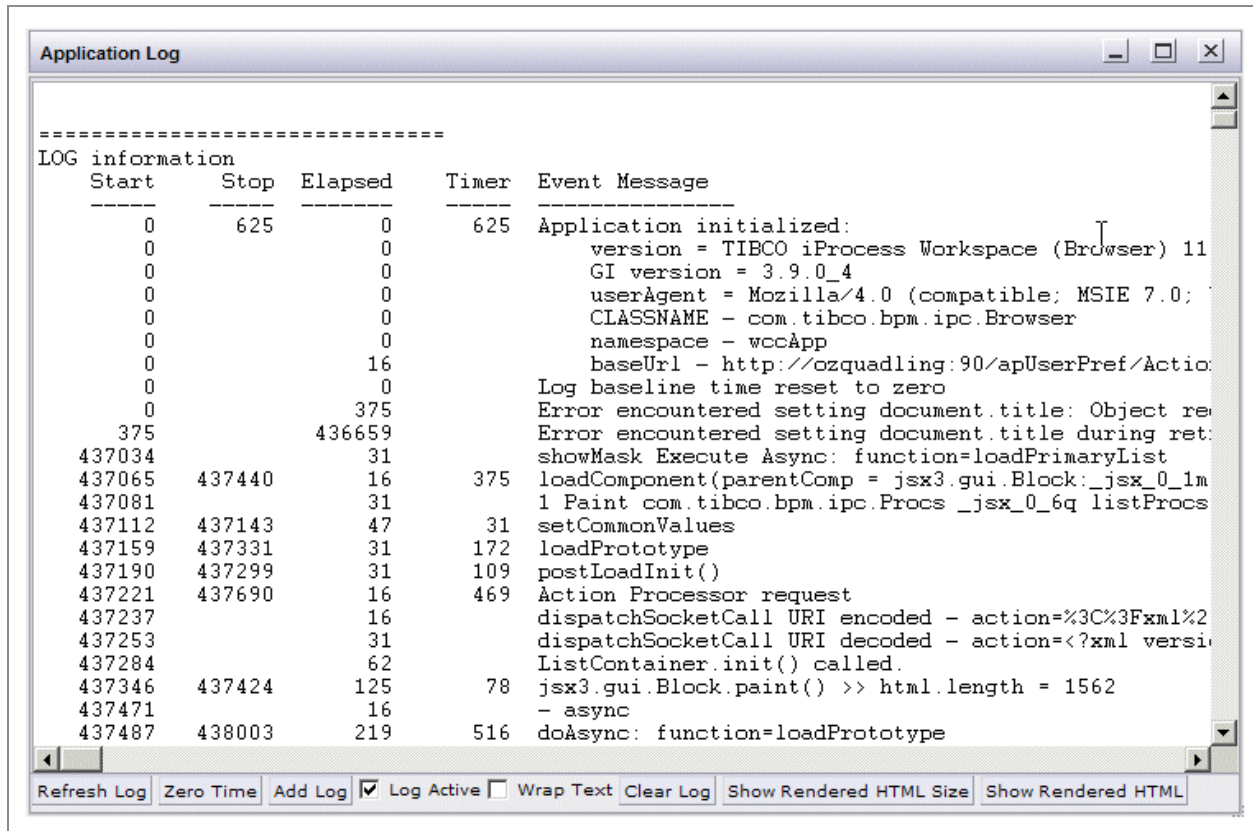
There are three types of TIBCO iProcess Workspace (Browser) logs available:

- **Session Activity Log** - This log allows users of the iProcess Workspace (Browser) to view information about activities they have performed in the application since they logged in. For information about this log, see the *TIBCO iProcess Workspace (Browser) User's Guide*.
- **Application Log** - This log provides detailed debug information, as well as communications between the client and the Action Processor. For more information, see [Application Log](#).
- **Application Monitor** - This log provides debug information on error conditions and exceptions encountered. For more information, see [Application Monitor](#).

Application Log

The Application Log is available to assist with troubleshooting the client application. This log provides detailed debug information generated by the iProcess Workspace (Browser), as well as information about communications between the client and Action Processor.

To display the Application Log, press the **F12** function key while the iProcess Workspace (Browser) is running. A window similar to the following is displayed:



Note that this log is available only if the logged-in user has “**ApplicationLog**” enabled in their user access profile. For more information, see [ApplicationLog](#).

Notice the **Log Active** check box on the bottom of the Application Log dialog. This box must be checked for the log to receive log data from the application.

Having the Application Log active can have an adverse effect on performance, therefore you can set the default state of the **Log Active** check box using the **appLogActive** attribute in the **logging** record in the application’s `config.xml` file.

The **logging** record also contains an attribute that allows you to echo the Application Log data to the Application Monitor — for information about the Application Monitor, see [Application Monitor](#).

To configure the Application Log:

1. Open the appropriate `config.xml` file, depending on whether you are configuring the iProcess Client or a custom application. For information about the file’s location, see [Configuration Files](#).
2. Locate the **logging** record in the `config.xml` file:

```
<record id="logging" type="ipc"
  appLogActive="false"
  echoToLog="false"/>
```


3. Modify the **appLogActive** attribute as follows:
 - “true” causes the **Log Active** check box in the Application Log to be checked by default.
 - “false” causes the **Log Active** check box in the Application Log to be unchecked by default.
4. Modify the **echoToJsxLog** attribute as follows:
 - “true” causes the contents of the Application Log to be echoed to the Application Monitor.
 - “false” causes the contents of the Application Log to not be echoed to the Application Monitor.

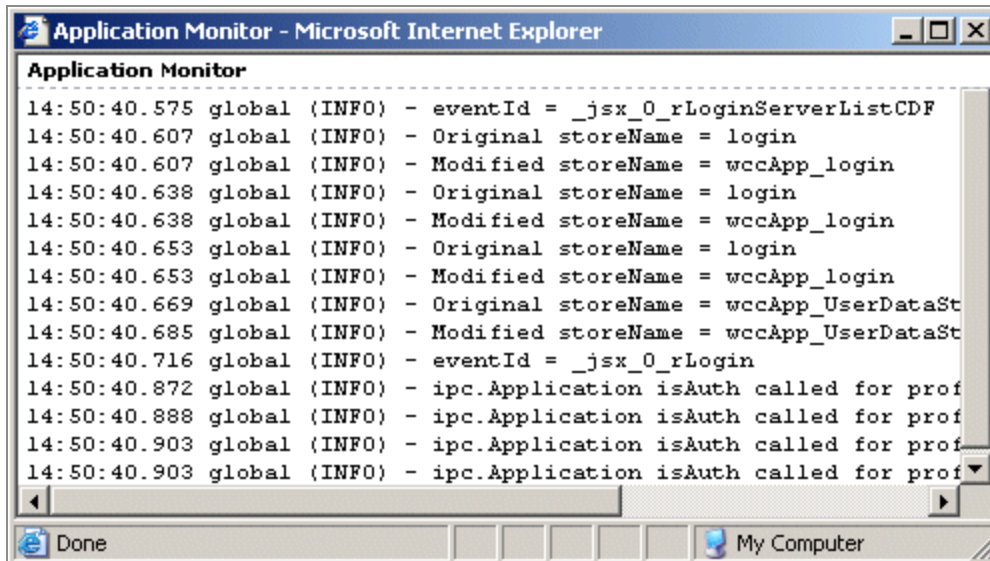
You can also use the buttons on the bottom of the Application Log dialog to do things such as clear and refresh the log, show rendered HTML, etc.

The Application Log can be closed by clicking in the X in the upper right corner of the Application Log dialog.

Application Monitor

The Application Monitor is available to assist with troubleshooting the client application. This monitor provides debug information on error conditions and exceptions encountered.

The Application Monitor is displayed in a separate browser window, which shows details of actions performed in the application. An example is shown below:



The Application Monitor can be configured using the following configuration file:

```
ClientInstallDir\logger.xml
```

where *ClientInstallDir* is the path to the directory in which the iProcess Workspace (Browser) is installed.

Default settings are specified by the following **handler** element in the **logger.xml** file, as shown below:

```
<handler name="ipcAppMonitorDefault" class="js3.app.Monitor" requires="true">
  <property name="serverNamespace" value="wccApp"/>
  <property name="disableIdle" eval="true" value="true"/>
  <property name="activateOnHotKey" eval="true" value="true"/>
</handler>
```

A reference to this handler is added under the global **logger** element:

```
<logger name="global" level="INFO">
  <handler-ref name="memory"/>
  ...
  <handler-ref name="ipcAppMonitorDefault"/>
</logger>
```

By default, both the Application Monitor and its hotkeys are enabled.

- To disable the Application Monitor, comment out the entire **<handler/>** element, as well as the **<handler-ref/>** element under the global **logger** element. (Note that if you comment out the Application Monitor, you must comment out both the **<handler/>** element, as well as the **<handler-ref/>** element. If the **<handler/>** element is commented out, but the **<handler-ref/>** element is not commented out, it results in a fatal error — the application will not load.)
- To disable the Application Monitor hotkeys, change the **activateOnHotKey** property's **value** attribute to "false".

When the Application Monitor's hotkeys are enabled, you can turn the monitor on and off using the **<Ctrl>+<Alt>+<m>** key sequence.

Also note that there are two logging categories used by the GI Forms add-in:

- form_adapter
- com.tibco.forms

To see log messages for these categories, add the following elements to the **logger.xml** file:

```
<logger name="form_adapter" level="INFO"/>  
<logger name="com.tibco.forms" level="INFO"/>
```

The level of the log messages can be set by changing the value of the **level** attribute in the **<logger name="global">** record. The valid levels are:

- FATAL
- ERROR
- WARN
- INFO
- DEBUG
- TRACE

You can also specify that Application Log data be echoed to the Application Monitor. This is accomplished using the **echoToJsxLog** attribute in the **logging** record in the application's **config.xml** file. For more information, see [Application Log](#).

Localization

This section describes how to add a language resource file to the iProcess Workspace (Browser) to display the application in the desired language.

Localizing the iProcess Workspace (Browser)

This section describes how to manually localize your TIBCO iProcess Workspace (Browser) client application.

Note that TIBCO has *language packs* available for selected languages, that when installed, localize your client application to the language for that language pack. For information about the available language packs, contact your TIBCO representative.

If the desired language is not available in a language pack, you can use the procedure described in this section to manually localize your client application.

Localizing the TIBCO iProcess Workspace (Browser) involves the following steps:

- Create a new localized language resource file. The resource file contains a collection of application text strings that have been translated to a specific language and may be localized for language variations used by individual countries.
- Configure the new localized language in the iProcess Workspace (Browser).
- Modify an existing, or create a new, TIBCO® General Interface system locale file. These files contain localized resources utilized by the TIBCO® General Interface framework.
- Translate user access profile descriptions.
- Set the new default language for the iProcess Workspace (Browser).
- Create a new folder to hold localized help files.

These steps are described in detail in the following subsections, using Spanish as an example of the new language being added to the iProcess Workspace (Browser).

Note that each localized language is represented by a two-letter code, in the format:

— ll

where ll is a lowercase, two-letter ISO 639 language code. For a list of language codes, visit the following web site:

<http://www.loc.gov/standards/iso639-2/langhome.html>

Each country is represented by a two-letter code, in the format:

— CC

where CC is an uppercase, two-letter ISO 3166 country code. For a list of country codes, visit the following web site:

<http://www.iso.ch/iso/en/prods-services/iso3166ma/02iso-3166-code-lists/index.html>

A locale key is a string representation of a locale that includes a language and a country code in the following format:

— ll_CC

Create a New Localized Language Resource File

You must create a resource file that contains a collection of application text strings that have been translated to a specific language and may be localized for language variations used by individual countries.

Perform the following steps to create a new language resource file:

1. Determine whether the new language file will contain translations that are:
 - Generic for all locales - for instance, Spanish is sufficient without regard to variations for the specific dialects or alphabets of Spain or Mexico.
 - Language defaults, with variations for specific locales - for instance, Spanish is the default, however, some words or phrases are defined specifically for the dialects or alphabets of Mexico and Spain.
 - Locale specific - for instance, if the Spanish of Mexico did not have any words or phrases in common with the Spanish of Spain, you would create a separate language resource file for each country.
2. Open a new XML file and insert the following XML elements, using the proper language code (if of type a or b above), or locale key containing both language and country codes (if of type c), as the value for the key attribute:

```
<data xmlns:space="propsbundle"
  <locale key="es">
</locale>
```

3. Open the default (English) locale file:

```
ClientInstallDir\JSXAPPS\ipc\locale\locale.xml
```

4. Copy all record elements that are direct children of the **<locale>** element in locale.xml.

Note - Only copy children of the **<locale>** elements that do *not* have a **key** attribute.

5. Paste all copied record elements into the newly created file as direct children of the **<locale key="es">** element.

```
<data xmlns:space="propsbundle"
  <locale key="es">
    <!-- PASTE ALL RECORD ELEMENTS HERE -->
```

6. Translate the value of every **jsxtext** attribute in the newly created file to language-specific values.

Note - Any record elements that are deleted from the new language resource file will cause the iProcess Workspace (Browser) to “fallback” to the record that is defined in the default (English) locale file.

7. Optionally, localize the new language resource for specific countries. The purpose of localizing for specific countries is to provide a mechanism for overriding default language text values (translated in Step 6) with text values that are specific for a country and that differ from the default (type b above).

For each country-specific locale, create a **<locale>** element (within the root **<data>** element) and specify the locale key as the value of the **key** attribute. Insert record elements into each new **<locale>** element, that are to “override” default language records, with matching **jsxid** attribute values.

```
<data xmlns:space="propsbundle"
  <locale key="es">
    <!-- DEFAULT LANGUAGE RECORD ELEMENTS HERE -->
    ...
    <record jsxid="txtClose" jsxtext="Cierre"/>
    <record jsxid="txtOpen" jsxtext="Abierto"/>
    ...
  </locale>
  <locale key="es_ES">
    <record jsxid="txtOpen" jsxtext="override value for Spain"/>
```

In the example above, the default Spanish language text of “Abierto” will be replaced with country-specific values when either the Spanish (Spain) or Spanish (Mexico) locales have been selected by the user as the language for the iProcess Workspace (Browser).

Any records not explicitly overridden in country-specific locales will “fallback” to the default language definition (e.g., “Cierre” in the example above).

8. Save the newly created locale resource file as follows:

```
ClientInstallDir\JSXAPPS\ipc\locale\locale.ll.xml
```

or (if of type c in step 1):

```
ClientInstallDir\JSXAPPS\ipc\locale\locale.ll_CC.xml
```

where ll in the filename is the language code, and CC is the country code (e.g., locale.es.xml - for Spanish; locale.es_MX.xml - for a Mexico-only translation) and *ClientInstallDir* is the path to the directory in which the iProcess Workspace (Browser) is installed.

Configure the New Localized Language in the iProcess Workspace (Browser)

Perform the following steps to configure the new localized language in the iProcess Workspace (Browser):

1. Add the new language code to the default language resource file.

Open *ClientInstallDir\JSXAPPS\ipc\locale\locale.xml* and edit the value of the **locales** attribute of the root **<data>** element.

Insert the two-letter language code of the new language into the comma-separated value of the **locales** attribute, as shown in the following example:

```
<data xmlns="http://www.tibco.com/xml/1.0" locales="default,es">
  <locale>
    <record jsid="and" jstext="AND"/>
  </locale>
</data>
```

Adding the language code to locale.xml provides the necessary configuration to support the “override” and “fallback” relationship between the new file and the default language resource file.

Modify or Create a TIBCO® General Interface System Locale File

Some of the text that is displayed in the iProcess Workspace (Browser) application originates from the TIBCO® General Interface (GI) system locale files. Although several formats and text strings are defined in these GI locale files, only a few text items will ever display in the iProcess Workspace (Browser).

By default, GI is already localized for many languages and countries, however, some of the text displayed in the iProcess Workspace (Browser) is only defined in the default GI system locale file (English), and must be inserted and translated into other locale files as required.

For this version of the iProcess Workspace (Browser), TIBCO® General Interface supports the following languages and countries:

Language	ISO 639-1 Code	ISO 639-1 Code + ISO 3166 Code	Country
Arabic	ar	N/A	
Chinese	zh	zh_CN	China
		zh_HK	Hong Kong
		zh_TW	Taiwan
Danish	da	da_DK	Denmark
English	en	en_AU	Australia
		en_CA	Canada
		en_GB	Great Britain (UK)
		en_NZ	New Zealand
		en_US	United States
French	fr	fr_CA	Canada
		fr_FR	France
German	de	de_DE	Germany
Greek	el	el_GR	Greece
Italian	it	it_IT	Italy
Japanese	ja	ja_JP	Japan

Language	ISO 639-1 Code	ISO 639-1 Code + ISO 3166 Code	Country
Korean	ko	ko_KR	Korea
Portuguese	pt	pt_BR	Brazil
		pt_PT	Portugal
Russian	ru	ru_RU	Russian Federation
Spanish	es	es_ES	Spain
		es_MX	Mexico
		es_US	United States
Swedish	sv	sv_SE	Sweden

The language resource files for these locales are stored at the following path:

ClientInstallDir\JSX\locale\

where *ClientInstallDir* is the path to the directory in which the iProcess Workspace (Browser) is installed. For example:

Folders		Name	Size	Type	Date Modified
3.9		locale.ar.xml	6 KB	XML Document	7/16/2010 3:39 PM
_uninst		locale.da.xml	6 KB	XML Document	7/16/2010 3:39 PM
addins		locale.de.xml	7 KB	XML Document	7/16/2010 3:39 PM
api		locale.el.xml	7 KB	XML Document	7/16/2010 3:39 PM
doc		locale.es.xml	8 KB	XML Document	7/16/2010 3:39 PM
GI		locale.fi.xml	8 KB	XML Document	7/16/2010 3:39 PM
GI_Builder		locale.fr.xml	8 KB	XML Document	7/16/2010 3:39 PM
GI_User		locale.he.xml	6 KB	XML Document	7/16/2010 3:39 PM
JSX		locale.id.xml	5 KB	XML Document	7/16/2010 3:39 PM
addins		locale.it.xml	6 KB	XML Document	7/16/2010 3:39 PM
css		locale.ja.xml	8 KB	XML Document	7/16/2010 3:39 PM
html		locale.ko.xml	7 KB	XML Document	7/16/2010 3:39 PM
images		locale.nl.xml	6 KB	XML Document	7/16/2010 3:39 PM
js		locale.nn.xml	5 KB	XML Document	7/16/2010 3:39 PM
jss		locale.pl.xml	6 KB	XML Document	7/16/2010 3:39 PM
locale		locale.pt.xml	9 KB	XML Document	7/16/2010 3:39 PM
sounds		locale.ru.xml	8 KB	XML Document	7/16/2010 3:39 PM
stubs		locale.sv.xml	6 KB	XML Document	7/16/2010 3:39 PM
xml		locale.th.xml	7 KB	XML Document	7/16/2010 3:39 PM
xsl		locale.tr.xml	6 KB	XML Document	7/16/2010 3:39 PM

As with the iProcess Workspace (Browser), the default GI language resource file is `locale.xml` (English). If the locale resource files already defined by TIBCO® General Interface do not support the desired localization being created for the iProcess Workspace (Browser), a new GI system locale file must be created using the following steps:

1. Create a new localized GI system language file using the same instructions as in [Create a New Localized Language Resource File](#) of this document, substituting `ClientInstallDir\JSX\locale\` as the file path.
2. Perform Step 1 of the section, [Configure the New Localized Language in the iProcess Workspace \(Browser\)](#) of this document, substituting `ClientInstallDir\JSX\locale\` as the file path.

If TIBCO® General Interface already provides a locale resource file for the desired localization, perform the following steps to add text resources, utilized by the iProcess Workspace, into the locale file:

3. Open the XML file corresponding to the language of the new localization (e.g., `ClientInstallDir\JSX\locale\locale.es.xml`) and open the default GI system locale file: `ClientInstallDir\JSX\locale\locale.xml`.
4. Copy the following record elements from `locale.xml` into the locale file of the new localization (under the **<locale>** element whose **key** attribute value matches the

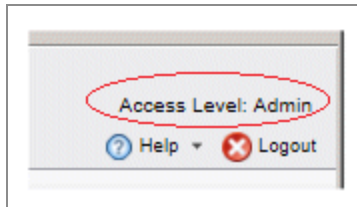
locale key of the new localization):

```
<record jsid="jsc3.gui.Select.defaultText" jsxtxt="Select" />
<record jsid="jsc3.gui.Select.dataUnavailable" jsxtxt="Data Unavailable" />
<record jsid="jsc3.gui.Select.noMatch" jsxtxt="No Match Found" />
<record jsid="jsc3.gui.Select.sel" jsxtxt="Selected" />
<record jsid="jsc3.gui.Menu.noData" jsxtxt="No Data" />
```

5. Translate the value of the **jsxtxt** attributes to the desired language.
6. Save the locale file (e.g., *ClientInstallDir\JSX\locale\locale.es.xml*).

Translate User Access Profiles Descriptions

User access profiles define the functionality available to various types of iProcess Workspace (Browser) users. The description of the profile defined for the logged-in user is displayed in the upper-right part of the application:



In the example above, “Access Level: Admin” is defined as the user access profile description for Admin-level users.

To localize user access profile descriptions, locate the **<Profile>** elements in the *ClientInstallDir\JSXAPPS\ipc\userAccessProfiles.xml* file (where *ClientInstallDir* is the path to the directory in which the iProcess Workspace (Browser) is installed), and change the **description** attributes to the desired language.

Set the New Default Language for the iProcess Workspace (Browser)

Set the new default language using one of the following methods:

- Modify the *config.xml* file to specify the default **localeKey**, as follows:

```
<record jsid="Options" type="ipc">
  <options>
    <display localeKey="en_US" initialDisplay="workQs"
      captionCases="name" captionWorkItems="name"
      autoRefreshWorkItems="true" autoRefreshInterval="60"
      autoRefreshApplyAll="true" />
  </options>
</record>
```

- Or, specify the language on the **Options** dialog in the application. For information about setting options in the application, see the *TIBCO iProcess Workspace (Browser) User's Guide*.

Create a New Folder to Hold Localized Help Files

If you have help files that have been localized, they need to be copied to the appropriate folder so that the iProcess Workspace (Browser) can find them based on the locale under which the application is running.

Perform the following steps to create the folders necessary to hold the localized help files for the iProcess Workspace (Browser).

1. Create a new folder at the following path:

`ClientInstallDir\Help\language\ll`

where *ClientInstallDir* is the path to the directory in which the iProcess Workspace (Browser) is installed, and ll is the two-letter language code for the help files.

If the help files are not country-specific and will be applicable for all locales of this particular language, proceed to Step 2 to store the files in this new language folder.

If your help files are localized for specific countries, create another folder beneath this new language folder, using the two-letter country code for the folder name (e.g. `ClientInstallDir\Help\language\ll\CC`). Proceed to Step 2 to store the files in this new country folder.

2. Copy the localized help files into the new folder.

IPC Tools Methods

This section describes tools methods available for iProcess Client applications.

Introduction

The IPC tools methods allow you to perform application functions through method calls when using a customized iProcess Client application, or from either custom GI Forms or TIBCO Forms used in an iProcess Client application. (Note that there are also equivalent methods called the "WCC" tools methods that can be used with a custom application created with WCC components; those are described in the *TIBCO iProcess Workspace (Browser) Components Reference* guide.

The IPC tools methods provide the same functionality that is available through the client application (as well as the WCC components), such as starting a case, opening a work item, triggering an event, etc.

Most of the IPC tools methods expect a *tag* of some sort (e.g., case tag, work item tag, etc.). Tags are intentionally opaque, that is, we do not provide the information needed to build them — you are expected to acquire them in one of the following ways:

- Tags can be acquired through the iProcess Server Objects object model, specifically using the **getTag** and **makeTag** methods. For information, see the *TIBCO iProcess Server Objects (Java or .NET) Programmer's Guide*.
- You can also acquire tags through an Action Processor response XML. For information, see the *iProcess Workspace (Browser) Action Processor Reference*.

These methods must be called in the context of the iProcess Client application object. For example, to call the **ipcStartCase** method, you would call either:

- `this.getApp().ipcStartCase(procTag);` (from within a javascript function)
- or —
- `com.tibco.bpm.ipc.getApp(this).ipcStartCase(procTag);` (from an event in a TIBCO® General Interface component)

A sample custom GI form that demonstrates the use of the IPC tools methods is also provided. For information about setting and using this sample, see [IPC Tools Methods Sample](#).

Method Summary

The following are the methods available:

- Case Functions
 - [ipcStartCase](#) - Starts a case of a procedure.
 - [ipcShowCase](#) - Displays information about a specific case.
 - [ipcCloseCases](#) - Closes one or more cases.
 - [ipcPurgeCases](#) - Purges (permanently deletes) one or more cases.
 - [ipcSuspendCases](#) - Suspends one or more cases.
 - [ipcActivateCases](#) - Reactivates one or more suspended cases.
 - [ipcShowGraphicalCaseHistory](#) - Displays the case history in a graphical format.
 - [ipcAddCaseHistoryEntry](#) - Adds an entry to the case history.
 - [ipcShowCasePrediction](#) - Predicts the outcome of the case.
 - [ipcTriggerEvent](#) - Starts process flow from an event step.
 - [ipcProcessJump](#) - Changes the process flow in a case.
- Work Item Functions
 - [ipcOpenWorkItem](#) - Opens (locks) a work item by passing in a work item tag.
 - [ipcOpenWorkItemEx](#) - Opens (locks) a work item by passing in parameters that identify the work item.
 - [ipcUnlockWorkItem](#) - Unlocks a work item.
 - [ipcForwardWorkItem](#) - Forwards a work item to a different work queue.
 - [ipcReleaseWorkItem](#) - Releases a work item.
- Work Queue Functions
 - [ipcConfigureSupervisors](#) - Allows set up of work queue supervisors.

- [ipcConfigureParticipation](#) - Allows set up of participation schedules.
- [ipcConfigureRedirection](#) - Allows set up of redirection schedules.
- [ipcShowWorkQLoadingChart](#) - Displays a graphical work queue summary.
- Procedure Functions
 - [ipcGetStartProcs](#) - Returns the procedures for which the logged-in user has permission to start cases.
 - [ipcGetAuditProcs](#) - Returns the procedures for which the logged-in user has permission to view or add entries to case history.
 - [ipcShowProcLoadingChart](#) - Displays a graphical procedure summary.
 - [ipcShowProcVersion](#) - Displays version history information for the procedure.
- Other Functions
 - [ipcShowServerInfo](#) - Displays information about the iProcess Objects Server.
 - [ipcShowOptions](#) - Displays interface for establishing default settings.
 - [ipcWorkItemTag2CaseTag](#) - Returns a case tag that is extracted from the work item tag passed as an argument.
 - [ipcWorkItemTag2WorkQTag](#) - Returns a work queue tag that is extracted from the work item tag passed as an argument.
 - [ipcGetUserAttributes](#) - Returns an array of objects that represent iProcess attributes assigned to a specific user.
 - [ipcGetGroupAttributes](#) - Returns an array of objects that represent iProcess attributes assigned to a specific group.
 - [ipcManageMER](#) - Displays the interface for managing Monitor Event Request (MER) messages on the server.

Login Required

All of the WCC methods require that the user be logged in prior to the method being called. A login can be accomplished in a number of ways, for example:

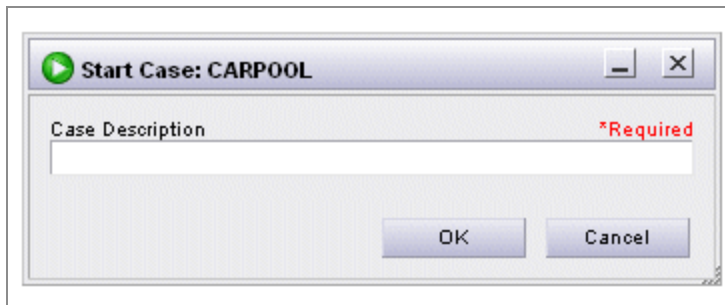
- using *single authentication*, which allows the user to be authenticated using credentials the user has already entered in another application — see the *TIBCO iProcess Workspace (Browser) Configuration and Customization* guide

- using the Action Processor **Login** request — see the *TIBCO iProcess Workspace (Browser) Action Processor Reference*.

ipcStartCase

This method starts a case of the specified procedure.

The **ipcStartCase** method displays the following dialog, which allows the user to enter a case description¹, then start the case by clicking on the **OK** button:



This method is equivalent to selecting **Start New Case** from the **Tools** menu on the procedure list in the iProcess Client application.

When the user clicks **OK**, a form is opened if the addressee of the first step in the procedure is SW_STARTER. The context in which the form is displayed depends on how the form was created, as follows:

- iProcess Modeler Forms - These forms are always displayed in a new browser window.
- TIBCO® General Interface Builder Forms - These forms (also known as “GI Forms”) are always displayed in a separate dialog.
- TIBCO Forms - These forms are always displayed in a separate dialog.

To start a case, the user must have permission to start cases of the particular procedure. To determine the procedures the user has permission to start, use the [ipcGetStartProcs](#) method.

¹The case description may or may not be required, depending on how the procedure was configured. If the case description is required, “*Required” is shown on the **Start Case** dialog.

Syntax

```
this.getApp().ipcStartCase(procTag);
```

Parameters

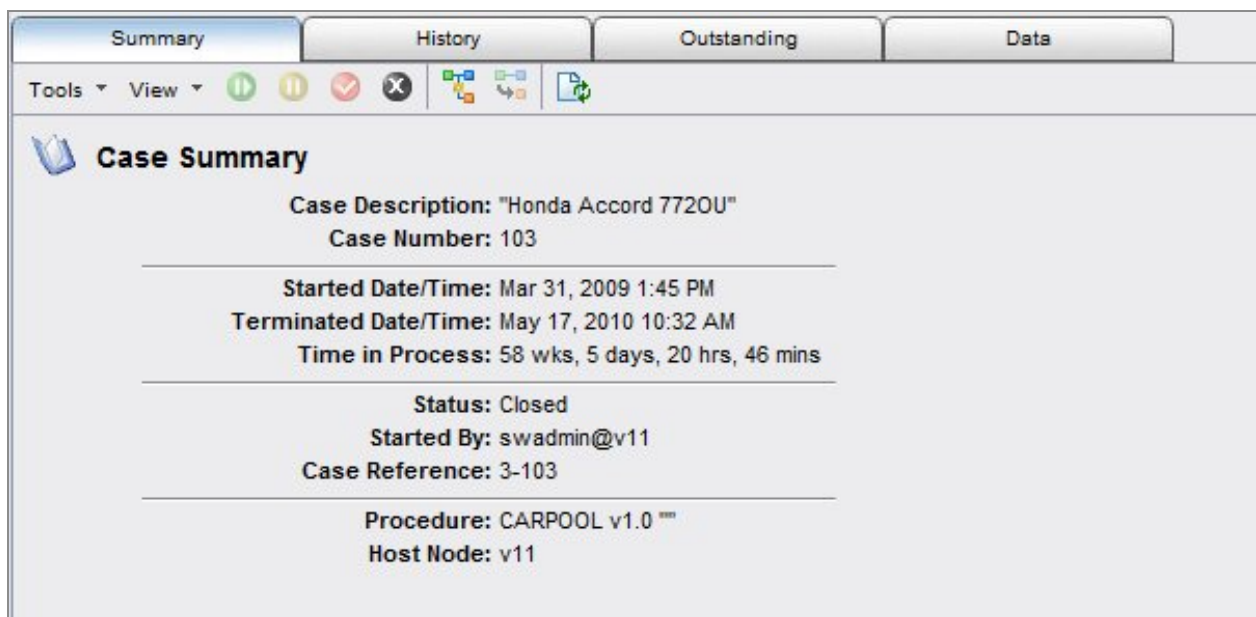
procTag - (String) Identifies the procedure for which you want to start a case. For information about tags, see [Introduction](#).

Example

```
this.getApp().ipcStartCase('i111|HIRING|1|0');
```

ipcShowCase

This method displays information, related to a specific case, in a dialog. The dialog contains four tabs of information, which may be optionally hidden using input parameters. For example:



Syntax

```
this.getApp().ipcShowCase(caseTag, hideSummary, hideHistory,
hideOutstanding, hideData);
```

Parameters

caseTag - (String) Identifies the case to display. For information about tags, see [Introduction](#).

hideSummary - (Boolean) If true, the **Summary** tab is removed from the dialog.

hideHistory - (Boolean) If true, the **History** tab is removed from the dialog. Note that to view the case history (that is, the **History** tab), the user must have permission to audit the procedure (of which the case is an instance). To determine the procedures the user has permission to audit, use the [ipcGetAuditProcs](#) method.

hideOutstanding - (Boolean) If true, the **Outstanding** tab is removed from the dialog.

hideData - (Boolean) If true, the **Data** tab is removed from the dialog.

Example

```
this.getApp().ipcShowCase('myserver|CARPOOL|0|1|1234', false, true, false,
false);
```

ipcCloseCases

This method closes the specified active cases of a procedure. This stops the process flow for the cases.

You must have system administrator authority to close cases.

An optional confirmation message can be displayed.

This method is equivalent to selecting **Close Case(s)** from the **Tools** menu on the case list in the iProcess Client application.

Syntax

```
this.getApp().ipcCloseCases(caseTags, suppressConfirm, doEvents);
```

Parameters

caseTags - (String or Array of Strings) Identifies the case(s) to close. For information about tags, see [Introduction](#).

suppressConfirm - (Boolean - Optional) Specifies whether or not to suppress the confirmation message. False (default) causes a confirmation message to be displayed; True suppresses the confirmation message.

doEvents - (Boolean - Optional) When true, a procedure-level event is triggered when the cases are closed. The procedure can be defined to catch the event, then perform business logic either before or after the case is closed. This parameter is only applicable when the *supressConfirm* parameter is true (when it is false, the user specifies this value on the confirmation dialog). This parameter is only relevant when an iProcess Engine version 11.4.0 or newer is being used. For earlier versions, this parameter is ignored. The default is true if not passed (and the engine version is 11.4.0 or newer).

Example

```
var caseTags = ["myserver|CARPOOL|0|1|1234", "myserver|CARPOOL|0|1|2342"];
this.getApp().ipcCloseCases(caseTags, true, true);
```

ipcPurgeCases

This method purges the specified cases of a procedure. Purging cases permanently deletes them from the system. You can purge both active and closed cases.

The user must have system administrator authority to purge cases.

An optional confirmation message can be displayed.

This method is equivalent to selecting **Purge Case(s)** from the **Tools** menu on the case list in the iProcess Workspace (Browser) client application.

Syntax

```
this.getApp().ipcPurgeCases(caseTags, suppressConfirm, doEvents);
```

Parameters

caseTags - (String or Array of Strings) Identifies the case(s) to purge. For information about tags, see [Introduction](#).

suppressConfirm - (Boolean - Optional) Specifies whether or not to suppress the confirmation message. False (default) causes a confirmation message to be displayed; True suppresses the confirmation message.

doEvents - (Boolean - Optional) When true, a procedure-level event is triggered when the cases are purged. The procedure can be defined to catch the event, then perform business logic before the cases are purged. This parameter is only applicable when the *supressConfirm* parameter is true (when it is false, the user specifies this value on the confirmation dialog). This parameter is only relevant when an iProcess Engine version 11.4.0 or newer is being used. For earlier versions, this parameter is ignored. The default is true if not passed (and the engine version is 11.4.0 or newer).

Example

```
var caseTags = ["myserver|CARPOOL|0|1|1234", "myserver|CARPOOL|0|1|2342"];
this.getApp().ipcPurgeCases(caseTags, true, true);
```

ipcSuspendCases

This method suspends one or more cases. Note that when you suspend a case, you are suspending the entire *case family*, which includes the main case and all of its sub-cases, if any.

When a case is suspended, current work items from that case can no longer be opened.

If a work item is already open when the case is suspended, the work item can still be kept, which causes the work item to become immediately suspended, and it cannot be opened again until the case is reactivated (see [ipcActivateCases](#)). The opened work item can also be released; this causes any new work items as a result of the release to become immediately suspended (unless they are flagged to ignore suspensions).

For more details about case suspensions, see the *TIBCO iProcess Workspace (Browser) User's Guide*.

An optional confirmation message can be displayed.

This method is equivalent to selecting **Suspend Case(s)** from the **Tools** menu on the case list in the iProcess Client application.

Syntax

```
this.getApp().ipcSuspendCases(caseTags, suppressConfirm);
```

Parameters

caseTags - (String or Array of Strings) Identifies the case(s) to suspend. For information about tags, see [Introduction](#).

suppressConfirm - (Boolean - Optional) Specifies whether or not to suppress the confirmation message. False (default) causes a confirmation message to be displayed; True suppresses the confirmation message.

Example

```
var caseTags = ["myserver|CARPOOL|0|1|1234", "myserver|CARPOOL|0|1|2342"];  
this.getApp().ipcSuspendCases(caseTags, true);
```

ipcActivateCases

This method reactivates one or more suspended cases (see [ipcSuspendCases](#)), which causes the process to flow as usual. Work items that were suspended because the case they are a part of was suspended can now be opened and processed normally.

An optional confirmation message can be displayed.

This method is equivalent to selecting **Activate Case(s)** from the **Tools** menu on the case list in the iProcess Client application.

Syntax

```
this.getApp().ipcActivateCases(caseTags, suppressConfirm);
```

Parameters

caseTags - (String or Array of Strings) Identifies the suspended case(s) to activate. For information about tags, see [Introduction](#).

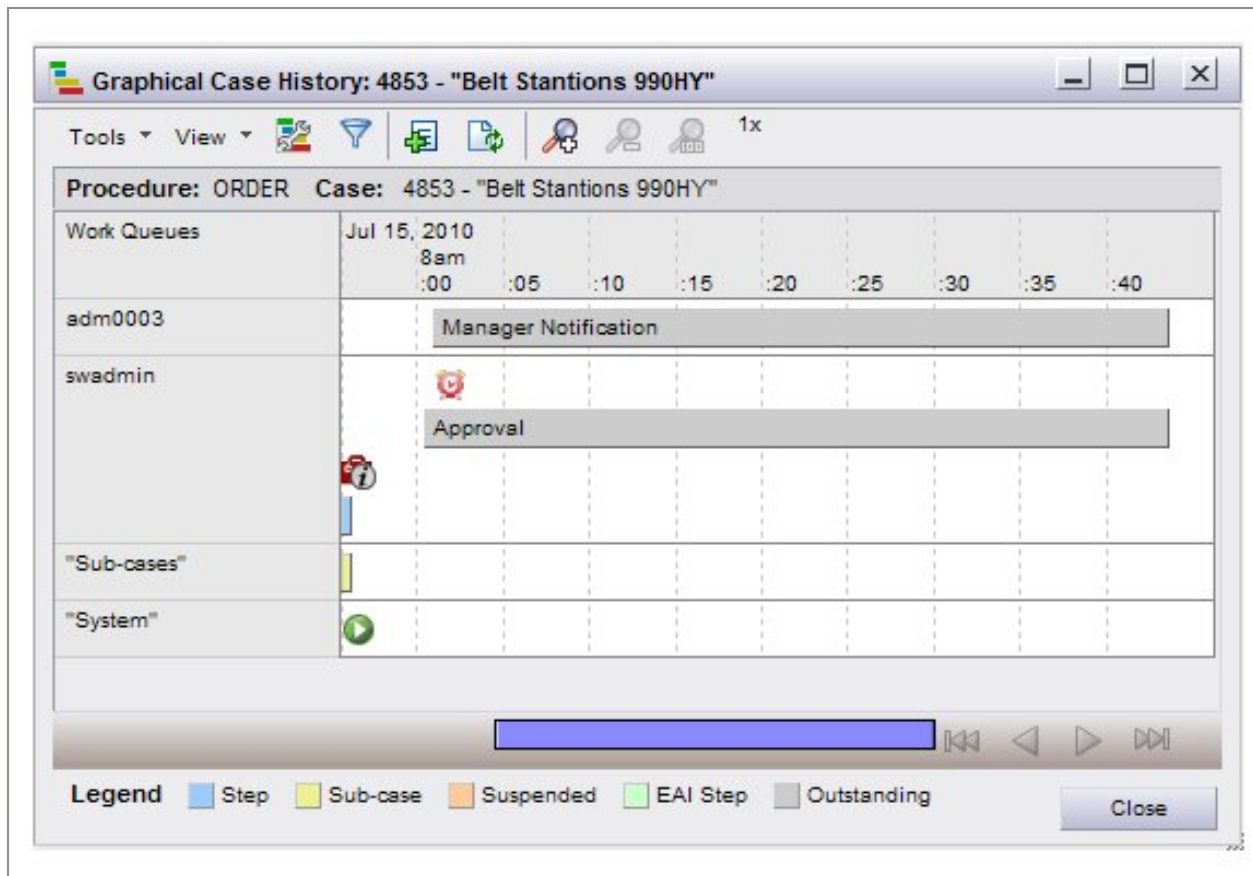
suppressConfirm - (Boolean - Optional) Specifies whether or not to suppress the confirmation message. False (default) causes a confirmation message to be displayed; True suppresses the confirmation message.

Example

```
var caseTags = ["myserver|CARPOOL|0|1|1234", "myserver|CARPOOL|0|1|2342"];  
this.getApp().ipcActivateCases(caseTags, true);
```

ipcShowGraphicalCaseHistory

This method displays the case history for the specified case in a graphical format. For example:



This method is equivalent to selecting **Graphical History** from the **View** menu on the case's **Summary** tab.

To view the graphical case history, the user must have permission to audit the procedure (of which the case is an instance). To determine the procedures the user has permission to audit, use the [ipcGetAuditProcs](#) method.

For more details about using the graphical case history, see the *TIBCO iProcess Workspace (Browser) User's Guide*.

Syntax

```
this.getApp().ipcShowGraphicalCaseHistory(caseTag);
```

Parameters

caseTag - (String) Identifies the case whose history to display in a graphical format. For information about tags, see [Introduction](#).

Example

```
this.getApp().ipcShowGraphicalCaseHistory('i2tagtest|CARPOOL|1|0|1851');
```

ipcAddCaseHistoryEntry

This method allows the user to manually add an entry to a case history.

This requires that a file (**auditusr.mes** file) be set up on the system that contains pre-defined messages that you can add to the case history.

This method causes a dialog similar to the following to be displayed:

The dialog box is titled "Add History: 4853 - \"Belt Stantions 990HY\"". It contains the following elements:

- Procedure:** ORDER **Case:** 4853 - \"Belt Stantions 990HY\"
- Select a template for history entry**
 - View** (dropdown menu) **3** (magnifying glass icon)

Number	Template
256	\"%DESC\" by %USER
257	%USER for \"%DESC\"
258	%USER has elevated %STEPNAME because \"%DESC\"
- Template Fields**
 - Description:** (%DESC)
 - Addressee:** (%USER)
 - Step Name:** (%STEPNAME)
- Buttons:** OK, Cancel

This dialog presents the messages that have been added to the **auditusr.mes** file. It allows the user to select which message to add to the case history.

**Note**

If your iProcess Engine does not support Add Case History “Templates”, the **Add History** dialog will contain a **Message Number** field instead of the list of available message numbers. If your system does not support templates, enter the the message number in the **Message Number** field.

This method is equivalent to selecting **Add Entry** from the **Tools** menu on the case’s **History** tab.

To add an entry to case history, the user must have permission to audit the procedure (of which the case is an instance). To determine the procedures the user has permission to audit, use the [ipcGetAuditProcs](#) method.

For more details about adding case history entries, see the *TIBCO iProcess Workspace (Browser) User’s Guide*.

Syntax

```
this.getApp().ipcAddCaseHistoryEntry(caseTag);
```

Parameters

caseTag - (String) Identifies the case to which you are adding an entry to case history. For information about tags, see [Introduction](#).

Example

```
this.getApp().ipcAddCaseHistoryEntry('i2tagtest|CARPOOL|1|0|1851');
```

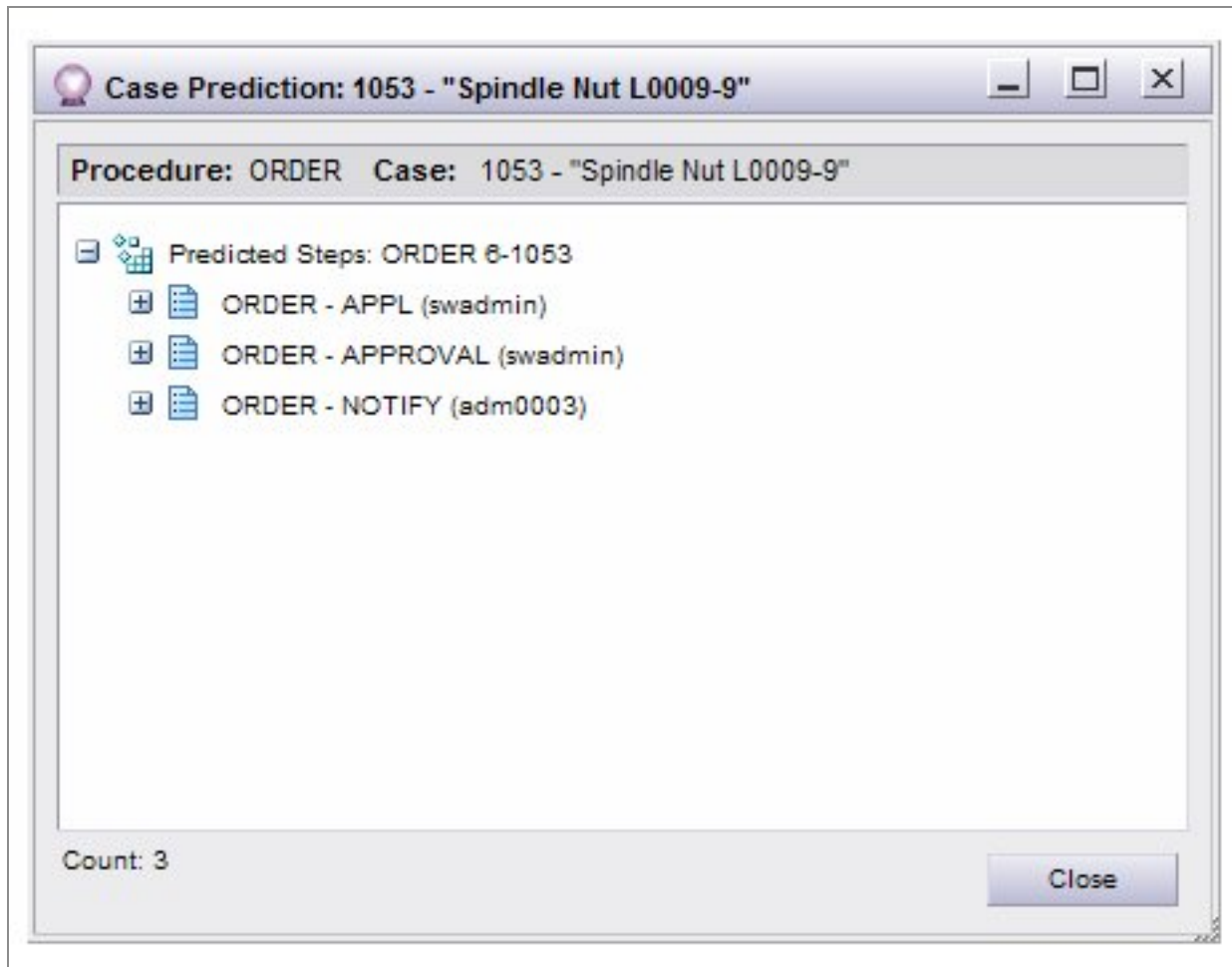
ipcShowCasePrediction

This method is used to predict the expected outcome of the specified live case. Running the case prediction function causes a list of “predicted work items” to be returned that

represent the work items that are currently due (outstanding work items), as well as the work items that are expected to be due in the future.

Included with the predicted work items returned is information about the expected times the work items are predicted to start and end, providing information that can be used to predict the outcome of the case. This can be used to improve work forecasting and estimate the expected completion of cases.

This method causes a dialog similar to the following to be displayed:



This provides a list of the *predicted steps* — the currently outstanding steps, and steps predicted to be outstanding as the case is processed to completion. For each step, it also indicates in parentheses the addressee of the step.

This method is equivalent to selecting **Predict Case** from the **Tools** menu on the case's **History** tab.

For more details about using case prediction, see the *TIBCO iProcess Workspace (Browser) User's Guide*.

Syntax

```
this.getApp().ipcShowCasePrediction(caseTag);
```

Parameters

caseTag - (String) Identifies the case on which you want to perform a case prediction function. For information about tags, see [Introduction](#).

Example

```
this.getApp().ipcShowCasePrediction('i2tagtest|CARPOOL|1|0|1851');
```

ipcTriggerEvent

This method is used to start the process flow from an *event step* in the procedure.

An event step is a step in a procedure that allows you to control the process flow in various ways, depending on how your procedure was designed. It can be used to perform actions such as:

- Suspending the flow of a case until an external action takes place.
- Starting a parallel branch in a case.
- Pausing a case for a specific period of time.

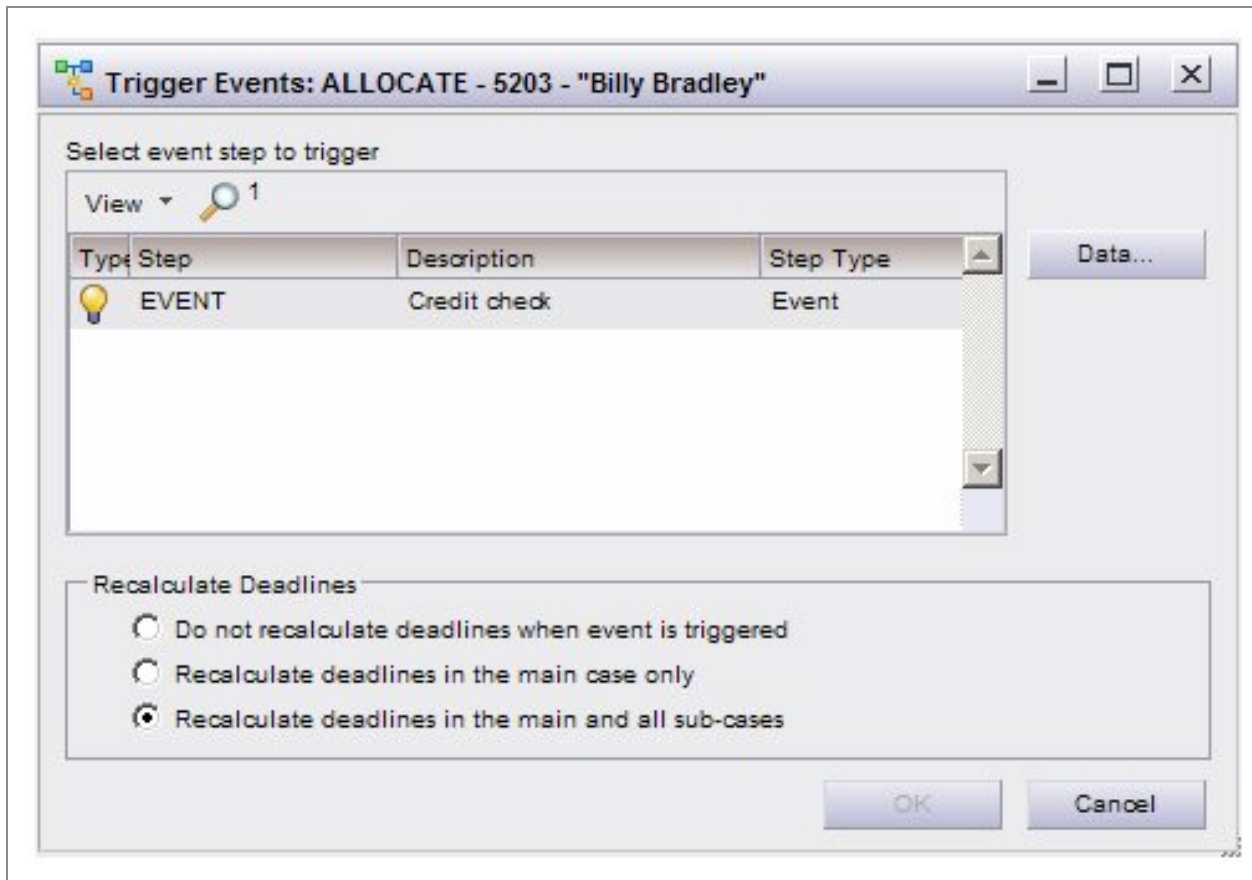
When the process flow reaches an event step, process flow is halted, and remains halted, until the user *triggers the event* with the **triggerEvent** method. When the event is triggered, the process flow will continue again.

However, an event step does not need to be outstanding (i.e., process flow has reached the step) to be triggered. You can trigger an event step at any time, such as:

- before the process flow has reached the event step,
- after the process flow has been halted at the event step, or
- after the event step has been triggered — one event step can be triggered multiple times. This allows you to run a segment of the procedure at any time, as many times as necessary.

Also note that other actions can be performed when an even step is triggered. These include resurrecting a closed case, as well as recalculating deadlines in the case. These actions and other details about triggering events are described in the *TIBCO iProcess Workspace (Browser) User's Guide*.

The **ipcTriggerEvent** method causes a dialog similar to the following to be displayed:



This dialog allows the user to select the event step from which the process flow should begin.

This method is equivalent to selecting **Trigger Event** from the **Tools** menu on the case's **Summary** tab.

Syntax

```
this.getApp().ipcTriggerEvent(caseTag);
```

Parameters

caseTag - (String) Identifies the case on which you want to trigger the event. For information about tags, see [Introduction](#).

Example

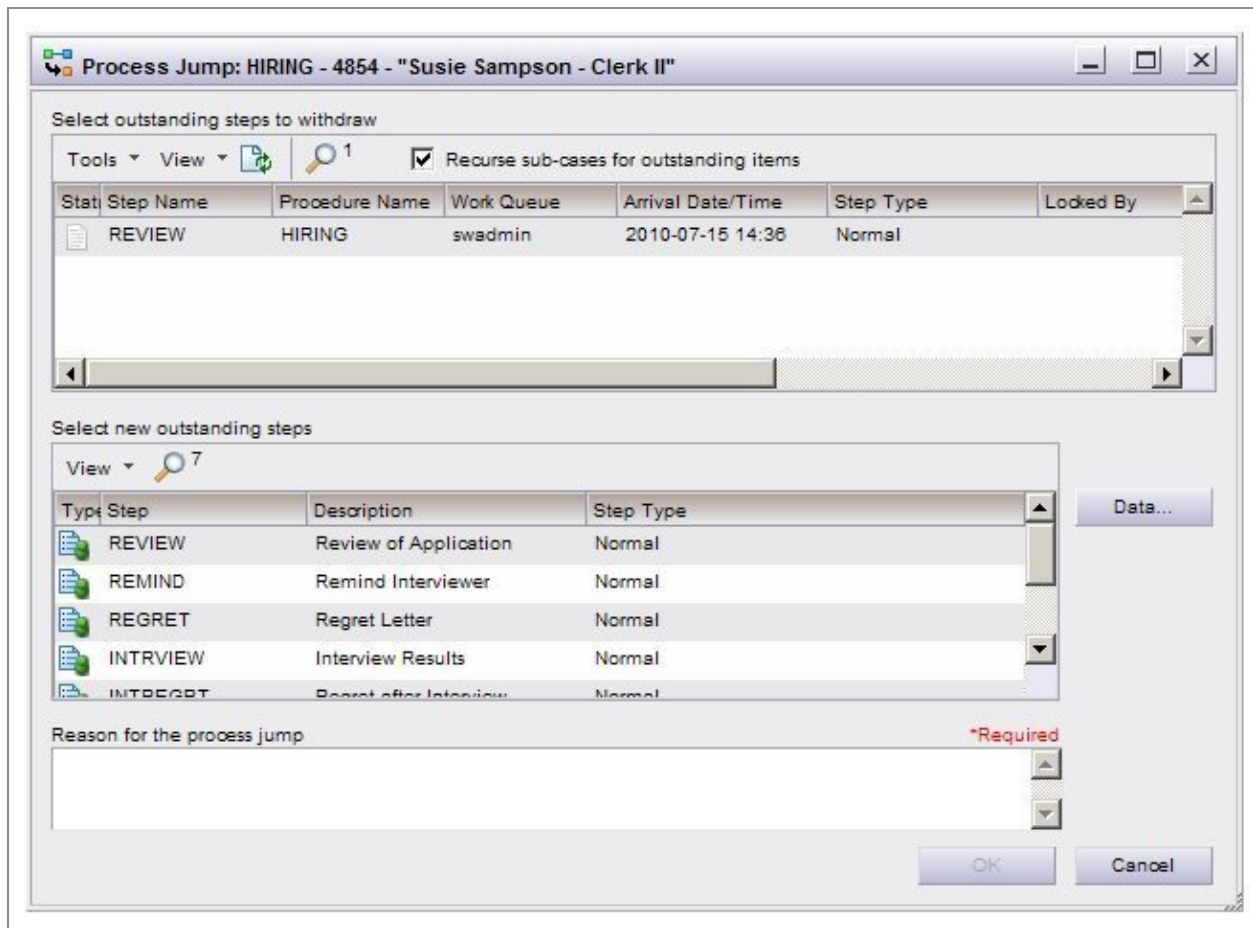
```
this.getApp().ipcTriggerEvent('i2tagtest|CARPOOL|1|0|1851');
```

ipcProcessJump

This method is used to change the process flow in the following ways:

- You can select currently outstanding steps you would like to “withdraw,” i.e., make them no longer outstanding.
- You can specify a set of steps to “jump to,” making those steps the new outstanding items.

Calling this method causes the following dialog to be displayed:



This dialog allows the user to select the outstanding steps to withdraw, as well as the new steps to make outstanding.

For more details about using the process jump function, see the *TIBCO iProcess Workspace (Browser) User's Guide*.

Syntax

```
this.getApp().ipcProcessJump(caseTag);
```

Parameters

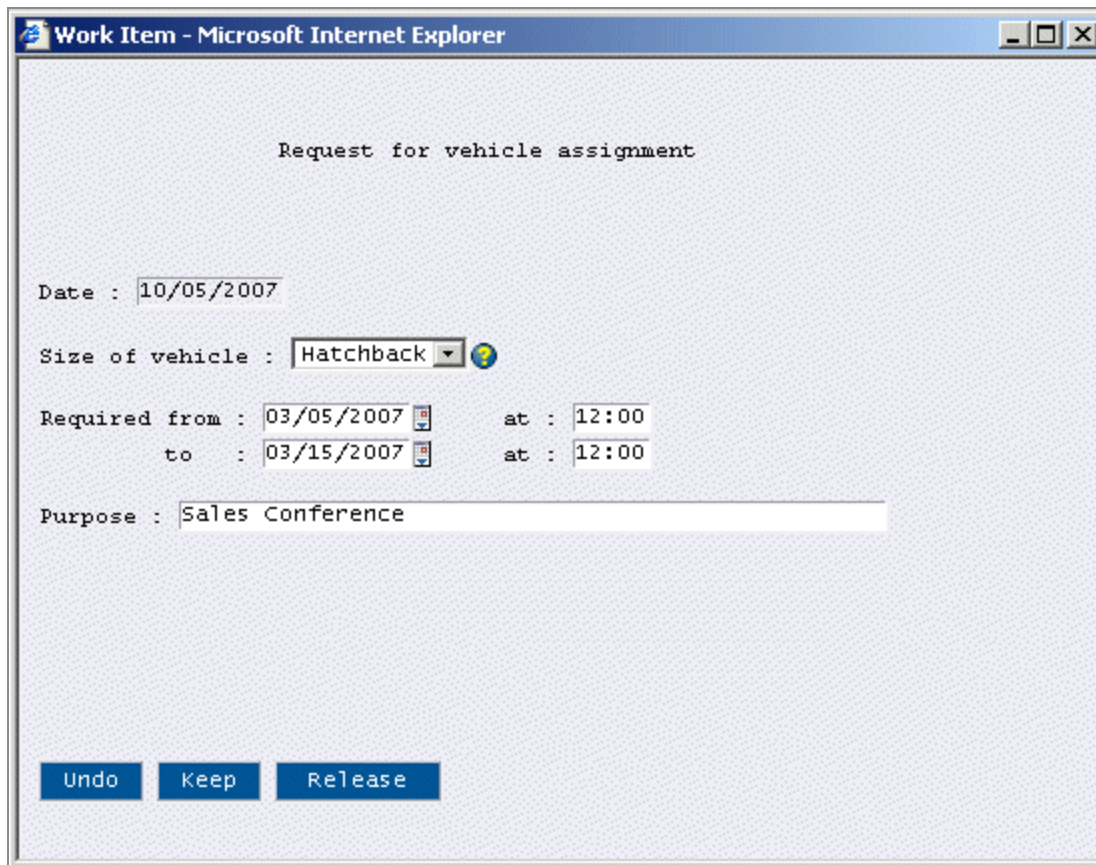
caseTag - (String) Identifies the case on which you want to change the process flow. For information about tags, see [Introduction](#).

Example

```
this.getApp().ipcProcessJump('i2tagtest|CARPOOL|1|0|1851');
```

ipcOpenWorkItem

This method opens (and locks) the specified work item and displays the form associated with that work item. For example:



Work Item - Microsoft Internet Explorer

Request for vehicle assignment

Date : 10/05/2007

Size of vehicle : Hatchback

Required from : 03/05/2007 at : 12:00
to : 03/15/2007 at : 12:00

Purpose : Sales Conference

Undo Keep Release

This method is equivalent to selecting **Open Selected Work Item(s)** from the **Tools** menu on the work item list in the iProcess Workspace (Browser) client application.

The context in which the form is displayed depends on how the form was created, as follows:

- iProcess Modeler Forms - These forms are always displayed in a new browser window (as in the example shown above).

- TIBCO® General Interface Builder Forms - These forms (also known as “GI forms”) are always displayed in a separate dialog.
- TIBCO Forms - These forms are always displayed in a separate dialog.

Also see the [ipcOpenWorkItemEx](#) method.

Syntax

```
this.getApp().ipcOpenWorkItem(workQTag, workItemTag);
```

Parameters

workQTag - (String) Identifies the work queue in which the work item resides. For information about tags, see [Introduction](#). (Note that a null can be passed in for the workQTag. If null, the workQTag is constructed from the workItemTag parameter.)

workItemTag - (String) Identifies the work item to open. For information about tags, see [Introduction](#).

Example

```
this.getApp().ipcOpenWorkItem  
( 'i11|swadmin|R', 'i11|S1|swadmin|R|1302|8152|i11|STEP1|0|0' );
```

ipcOpenWorkItemEx

This method opens (and locks) the specified work item and displays the form associated with that work item. Unlike the **ipcOpenWorkItem** method (see [ipcOpenWorkItem](#)), however, you do not pass in a work item tag with this method — instead, the work item tag is obtained from a list of outstanding work items using the information passed in the parameters.

An example work item form is shown below:

Request for vehicle assignment

Date : 10/05/2007

Size of vehicle : Hatchback

Required from : 03/05/2007 at 12:00
to : 03/15/2007 at 12:00

Purpose : Sales Conference

Undo Keep Release

This method is equivalent to selecting **Open Selected Work Item(s)** from the **Tools** menu on the work item list in the iProcess Client application.

The context in which the form is displayed depends on how the form was created, as follows:

- iProcess Modeler Forms - These forms are always displayed in a new browser window (as in the example shown above).
- TIBCO® General Interface Builder Forms - These forms (also known as “GI forms”) are always displayed in a separate dialog.
- TIBCO Forms - These forms are always displayed in a separate dialog.

Syntax

```
this.getApp().ipcOpenWorkItemEx(caseNumber, procName, stepName, queueName, queueReleased);
```

Parameters

caseNumber - (String) Identifies the case in which the work item was created.

procName - (String) Identifies the procedure.

stepName - (String) Identifies the step in the procedure that corresponds to the work item.

queueName - (String) Identifies the work queue in which the work item resides.

queueReleased - (String) "Y" or "N" indicating whether or not the work queue is released.

Example

```
this.getApp().ipcOpenWorkItemEx('13502', 'ONESTEP', 'STEP1', 'swadmin', 'Y');
```

ipcUnlockWorkItem

This method unlocks the specified work item.

This method is equivalent to selecting **Unlock Work Item(s)** from the **Tools** menu on the work item list in the iProcess Client application.

Note that work items are automatically unlocked when you keep or release them; normally, you do not need to explicitly unlock work items. This function is for those rare occasions when a work item was left open for some reason (e.g., a system crash).

Unlocking a work item using this method causes any changes that were made on the form while the work item was open to be discarded.

Any user can unlock a work item that they have opened. To unlock a work item that was opened by another user, you must have system administrator authority.

If you attempt to unlock a work item that is not locked, the method call returns silently with no error.

An optional confirmation message can be displayed.

Syntax

```
this.getApp().ipcUnlockWorkItem(workQTag, workItemTag, suppressConfirm);
```

Parameters

workQTag - (String) Identifies the work queue in which the work item resides. For information about tags, see [Introduction](#). (Note that a null can be passed in for the workQTag. If null, the workQTag is constructed from the workItemTag parameter.)

workItemTag - (String) Identifies the work item to unlock. For information about tags, see [Introduction](#).

suppressConfirm - (Boolean - Optional) Specifies whether or not to suppress the confirmation message. False (default) causes a confirmation message to be displayed; True suppresses the confirmation message.

Example

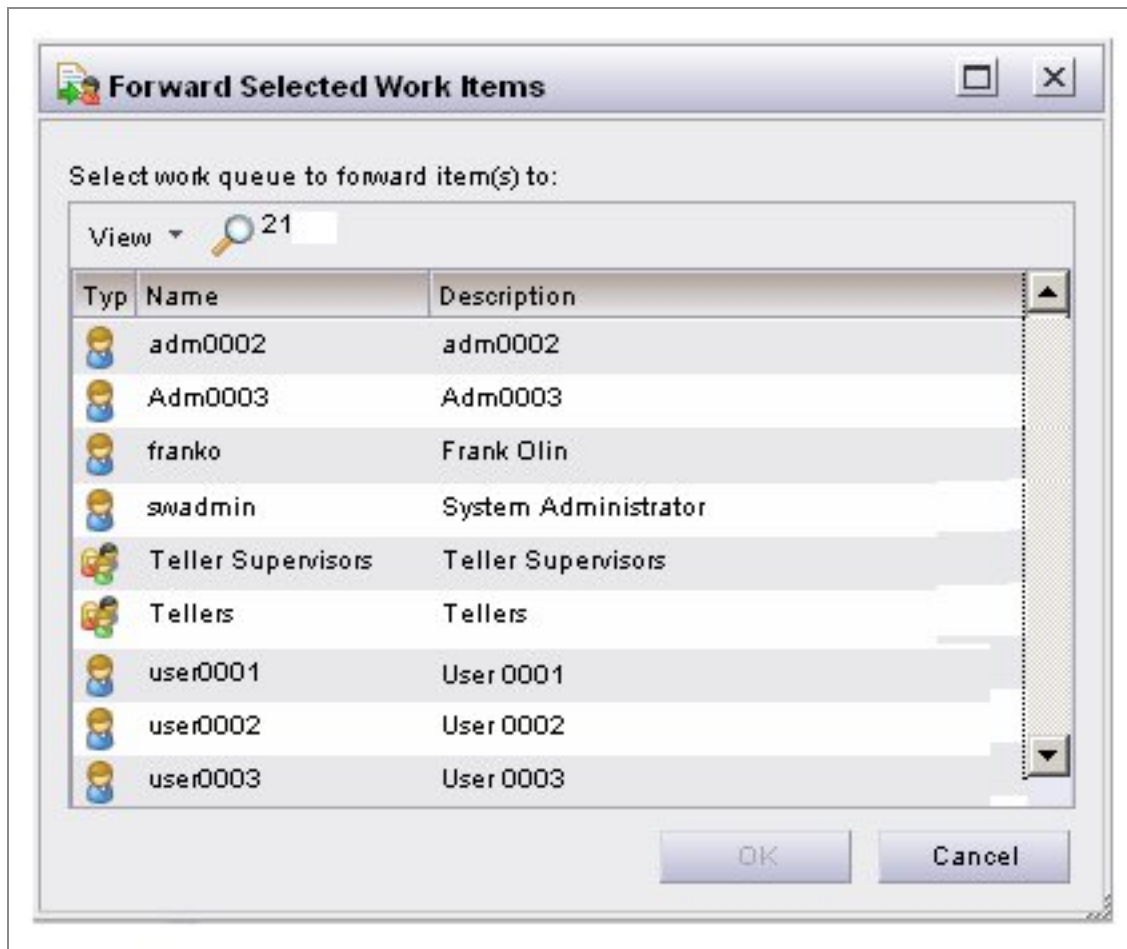
```
this.getApp().ipcUnlockWorkItem('i11|smith|R',  
'i11|S1|smith|R|135|812|i11|SP1|0|0', true);
```

ipcForwardWorkItem

This method forwards the specified work item to a different work queue.

Note that not all work items are “forwardable.” When a procedure is defined, the designer specifies whether or not work items representing each step are forwardable. (There is a “Forwardable” column available on the work item list that indicates whether or not a work item is forwardable.)

This method causes a dialog similar to the following to be displayed:



This dialog will list either all work queues on the system, or only the work queues of which the user is a member, depending on how the user's user access profile is set up.

The user selects the desired work queue from the list, then clicks **OK**. The work item specified in the method call is forwarded.

This method is equivalent to selecting **Forward Work Item(s)** from the **Tools** menu on the work item list in the iProcess Workspace (Browser) client application.

Syntax

```
this.getApp().ipcForwardWorkItem(workQTag, workItemTag);
```

Parameters

workQTag - (String) Identifies the work queue in which the work item being forwarded currently resides. For information about tags, see [Introduction](#). (Note that a null can be passed in for the workQTag. If null, the workQTag is constructed from the workItemTag parameter.)

workItemTag - (String) Identifies the work item to forward. For information about tags, see [Introduction](#).

Example

```
this.getApp().ipcForwardWorkItem  
( 'i11|swadmin|R', 'i11|S1|smith|R|132|8152|i11|STEP1|0|0' );
```

ipcReleaseWorkItem

This method releases the specified work item.

Note that you can only release work items with this method that are considered “directly releasable”, i.e., *they do not have any input fields on their form* (if they have a form). That does not mean their input fields have been filled in — they cannot have input fields. (There is a **Releasable** column available on the work item list that indicates whether or not a work item is directly releasable.)

Releasing the work item causes the case to advance to the next step in the procedure, possibly resulting in another work item appearing in someone’s work queue.

This method is equivalent to selecting **Release Work Item(s)** from the **Tools** menu on the work item list in the iProcess Client application.

Syntax

```
this.getApp().ipcReleaseWorkItem(workQTag, workItemTag);
```

Parameters

workQTag - (String) Identifies the work queue in which the work item being released currently resides. For information about tags, see [Introduction](#). (Note that a null can be passed in for the workQTag. If null, the workQTag is constructed from the workItemTag parameter.)

workItemTag - (String) Identifies the work item to release. For information about tags, see [Introduction](#).

Example

```
this.getApp().ipcReleaseWorkItem  
( 'i11|swadmin|R', 'i11|S1|smith|R|132|8152|i11|STEP1|0|0' );
```

ipcConfigureSupervisors

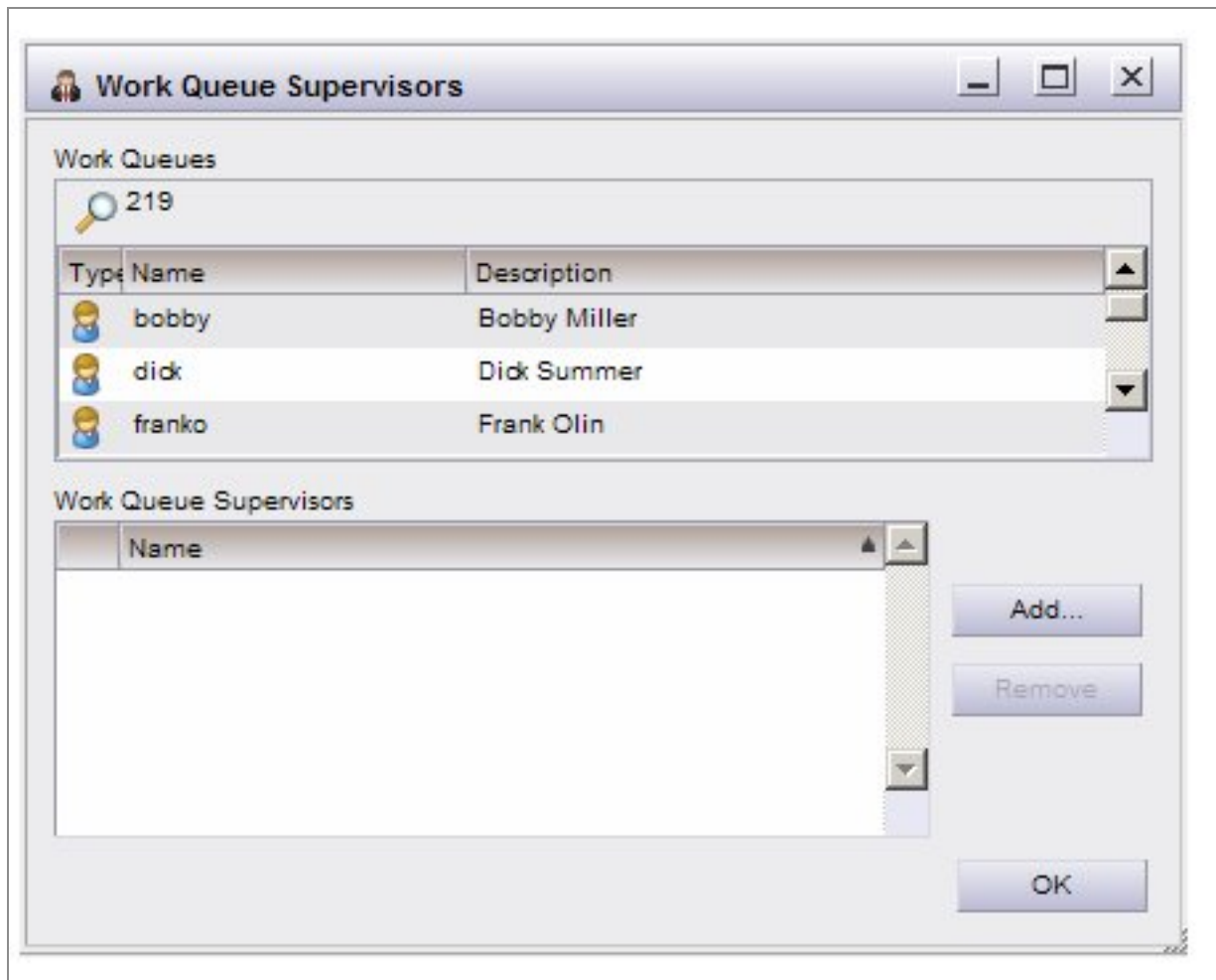
This method is used to designate users as work queue supervisors. A user must be a work queue supervisor to perform the following tasks:

- Configure **participation schedules**. A participation schedule gives another user temporary access to a work queue. For information about configuring participation schedules, see [ipcConfigureParticipation](#).
- Configure **redirection schedules**. A redirection schedule causes work items to be temporarily redirected to another work queue. For information about configuring redirection schedules, see [ipcConfigureRedirection](#).

Each work queue can be assigned one or more work queue supervisors.

You must have system administrator authority to configure work queue supervisors.

This method causes a dialog similar to the following to be displayed:



The **Work Queues** section of this dialog lists all work queues (i.e., all users and groups) defined on your TIBCO system. The **WorkQueue Supervisors** section lists the supervisors for the currently selected work queue.

For more details about configuring work queue supervisors, see the *TIBCO iProcess Workspace (Browser) User's Guide*.

This method is equivalent to selecting **Manage Work Queue Supervisors** from the **Tools** menu on the work queue list.

Syntax

```
this.getApp().ipcConfigureSupervisors();
```

Parameters

None

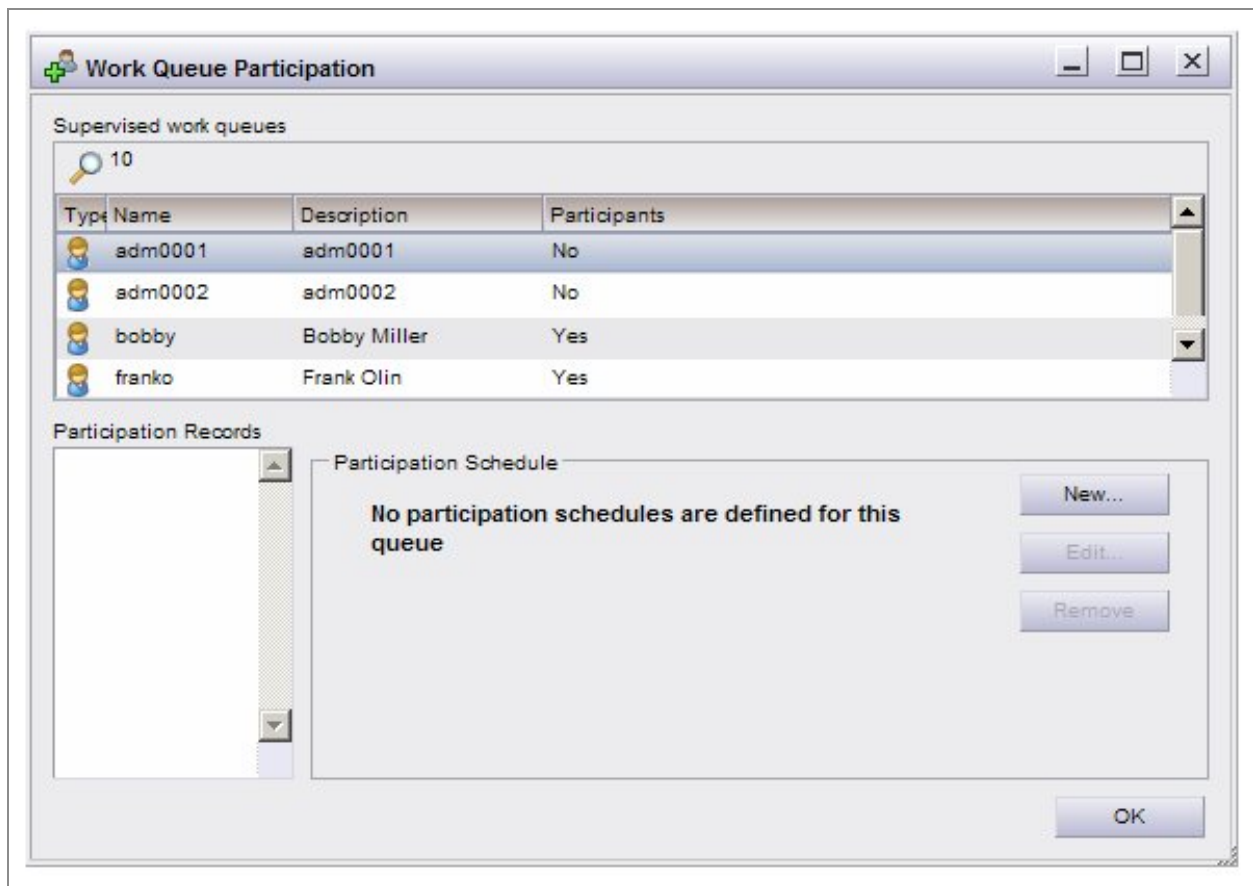
Example

```
this.getApp().ipcConfigureSupervisors();
```

ipcConfigureParticipation

This method is used to configure *participation schedules*, which specify that a user can *participate* in (i.e., have access to) another user's work queue for a specified period of time.

This method causes a dialog similar to the following to be displayed:



The **Supervised work queues** section lists all work queues for which the user has been designated a supervisor — these are the work queues for which the user is authorized to configure participation schedules. (For information about designating a user a work queue supervisor, see [ipcConfigureSupervisors](#).)

For details about work queue participation, see the *TIBCO iProcess Workspace (Browser) User's Guide*.

This method is equivalent to selecting **Manage Work Queue Participation** from the **Tools** menu on the work queue list.

Syntax

```
this.getApp().ipcConfigureParticipation();
```

Parameters

None

Example

```
this.getApp().ipcConfigureParticipation();
```

ipcConfigureRedirection

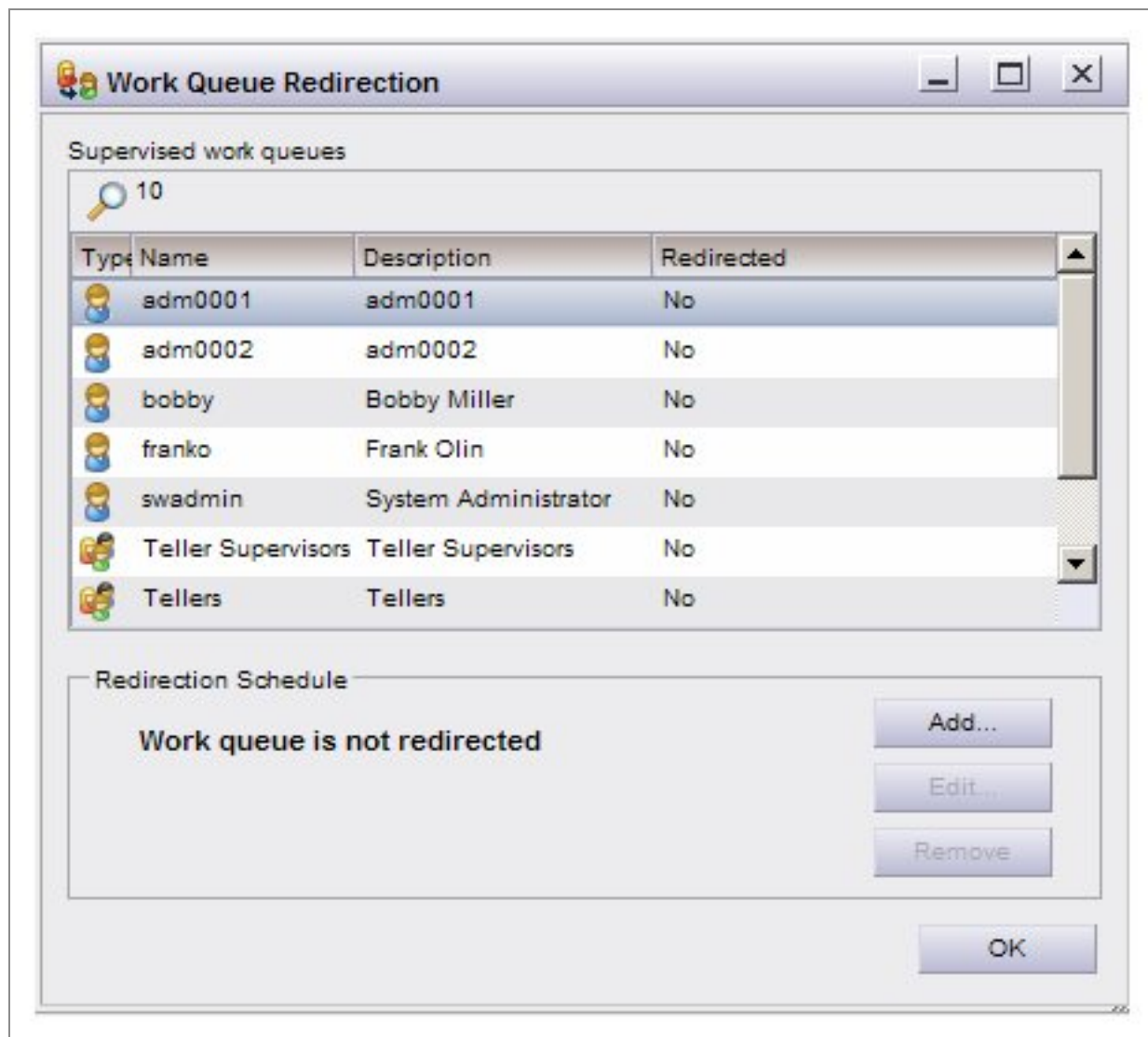
This method is used to configure *redirection schedules*, which are used to *redirect* one user's work items to the work queue of another user or group for a specified period of time.



Note

For information about forwarding an individual work item from a work queue, see [ipcForwardWorkItem](#).

This method causes a dialog similar to the following to be displayed:



The **Supervised work queues** section lists all work queues for which the user has been designated a supervisor — these are the work queues for which the user is authorized to configure redirection schedules. (For information about designating a user a work queue supervisor, see [ipcConfigureSupervisors](#).)

For details about configuring redirection schedules, see the *TIBCO iProcess Workspace (Browser) User's Guide*.

This method is equivalent to selecting **Manage Work Queue Redirection** from the **Tools** menu on the work queue list.

Syntax

```
this.getApp().ipcConfigureRedirection();
```

Parameters

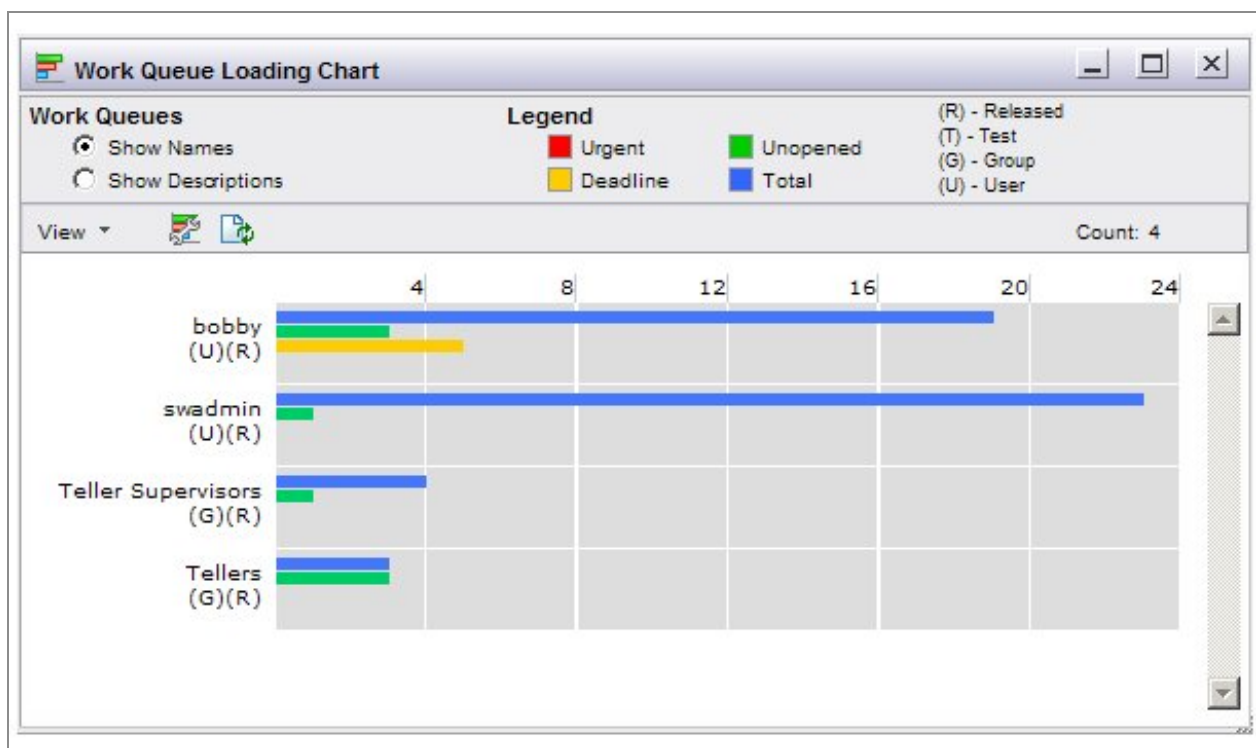
None

Example

```
this.getApp().ipcConfigureRedirection();
```

ipcShowWorkQLoadingChart

This method displays a graphical summary of the work queues available to the user. For example:



This chart provides information about the numbers and types of work items in each work queue.

This method is equivalent to selecting **Work Queue Loading Chart** from the **Tools** menu on the work queue list.

Syntax

```
this.getApp().ipcShowWorkQLoadingChart(releasedQs, testQs, groupQs, userQs);
```

Parameters

releasedQs - (Boolean - Optional) Include released work queues in loading chart? Default = True.

testQs - (Boolean - Optional) Include test work queues in loading chart? Default = True.



Note

You must pass True for either *releasedQs* or *testQs*. If you pass False for both parameters, loading information is displayed for both released and test work queues.

groupQs - (Boolean - Optional) Include group work queues in loading chart? Default = True.

userQs - (Boolean - Optional) Include user work queues in loading chart? Default = True.



Note

You must pass True for either *groupQs* or *userQs*. If you pass False for both parameters, loading information is displayed for both group and user work queues.

Example

```
this.getApp().ipcShowWorkQLoadingChart(true, false, false, false);
```

ipcGetStartProcs

This method returns an array of JavaScript objects that represent iProcess procedures for which the logged-in user has permission to start cases.

Each JavaScript object returned represents a procedure, and has a Name, Description, HostingNode, MajorVersion, MinorVersion, and Tag property.

This method can be used prior to calling the [ipcStartCase](#) method to determine the procedures the logged-in user can start.

Syntax

```
this.getApp().ipcGetStartProcs();
```

Parameters

None

Example

```
var startProcs = this.getApp().ipcGetStartProcs();
var msg = 'Name/Description/HostingNode/MajorVersion/MinorVersion/Tag' +
'\n';
for (var i = 0; i < startProcs.length; i++) {
    msg += startProcs[i].Name + '/' +
        startProcs[i].Description + '/' +
        startProcs[i].HostingNode + '/' +
        startProcs[i].MajorVersion + '/' +
        startProcs[i].MinorVersion + '/' +
        startProcs[i].Tag + '"\n';
}
alert(msg);
```

ipcGetAuditProcs

This method returns an array of JavaScript objects that represent iProcess procedures for which the logged-in user has permission to view or add entries to case history.

Each JavaScript object returned represents a procedure, and has a Name, Description, HostingNode, MajorVersion, MinorVersion, and Tag property.

This method can be used prior to calling the [ipcShowGraphicalCaseHistory](#) or [ipcAddCaseHistoryEntry](#) method to determine if the logged-in user has permission to view the graphical case history or add an entry to case history. It can also be used prior to calling the [ipcShowCase](#) method to determine if the logged-in user can view the **History** tab when displaying information about a case.

Syntax

```
this.getApp().ipcGetAuditProcs();
```

Parameters

None

Example

```
var auditProcs = this.getApp().ipcGetAuditProcs();
var msg = 'Name/Description/HostingNode/MajorVersion/MinorVersion/Tag' +
'\n';
for (var i = 0; i < auditProcs.length; i++) {
    msg += auditProcs[i].Name + '/' +
        auditProcs[i].Description + '/' +
        auditProcs[i].HostingNode + '/' +
        auditProcs[i].MajorVersion + '/' +
        auditProcs[i].MinorVersion + '/' +
        auditProcs[i].Tag + '"\n';
}
alert(msg);
```

ipcShowProcLoadingChart

This method displays a graphical summary of the procedures on the system. For example:



This chart provides information about the numbers and types of cases that exist for each procedure.

This method is equivalent to selecting **Procedure Loading Chart** from the **Tools** menu on the procedure list.

Syntax

```
this.getApp().ipcShowProcLoadingChart(released, unreleased, model,
withdrawn);
```

Parameters

released - (Boolean - Optional) Include released procedures in loading chart? Default = True.

unreleased - (Boolean - Optional) Include unreleased procedures in loading chart? Default = True.

model - (Boolean - Optional) Include model procedures in loading chart? Default = True.

withdrawn - (Boolean - Optional) Include withdrawn procedures in loading chart?
Default = True.

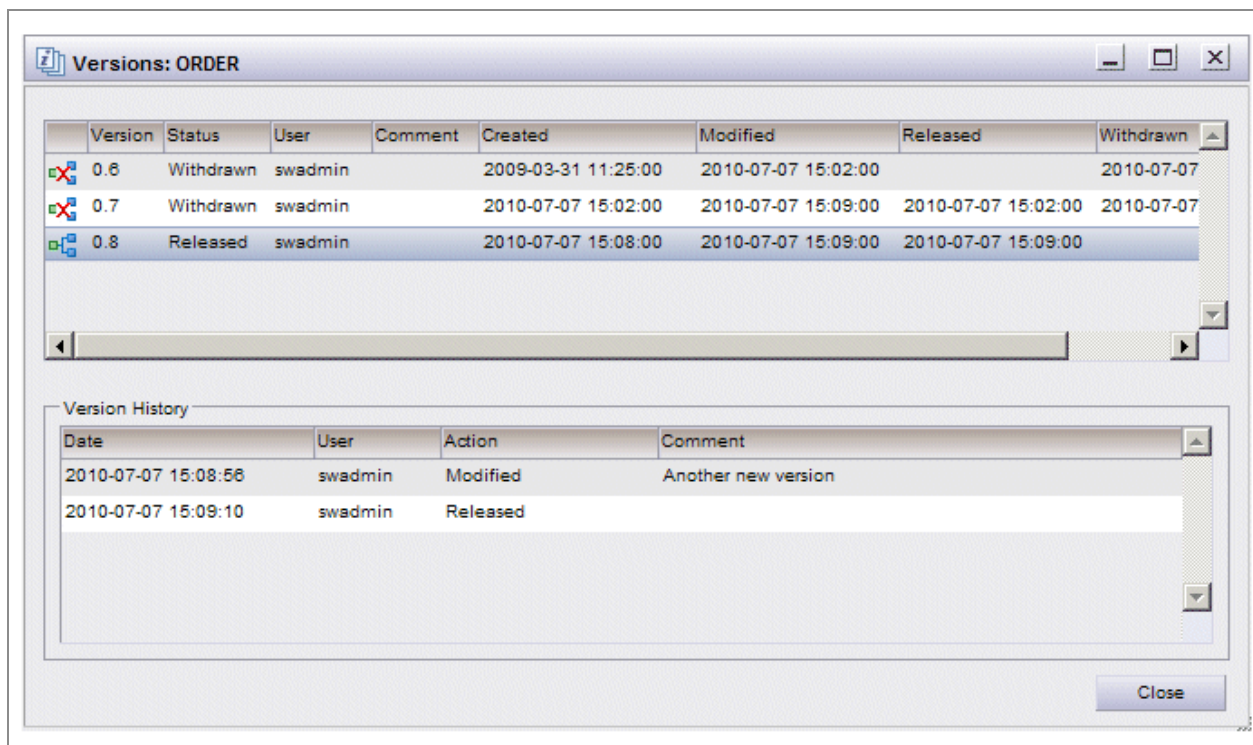
Example

```
this.getApp().ipcShowProcLoadingChart(true, false, false, false);
```

ipcShowProcVersion

This method displays information about the past and current versions of the specified procedure.

This method displays a dialog similar to the following:



Clicking on one of the versions in the top section causes history information about that version to be displayed in the section on the bottom of the dialog.

This method is equivalent to selecting **Versions** from the **Tools** menu on the procedure list.

Syntax

```
this.getApp().ipcShowProcVersion(procTag);
```

Parameters

procTag - (String) Identifies the procedure whose version information you want displayed. For information about tags, see [Introduction](#).

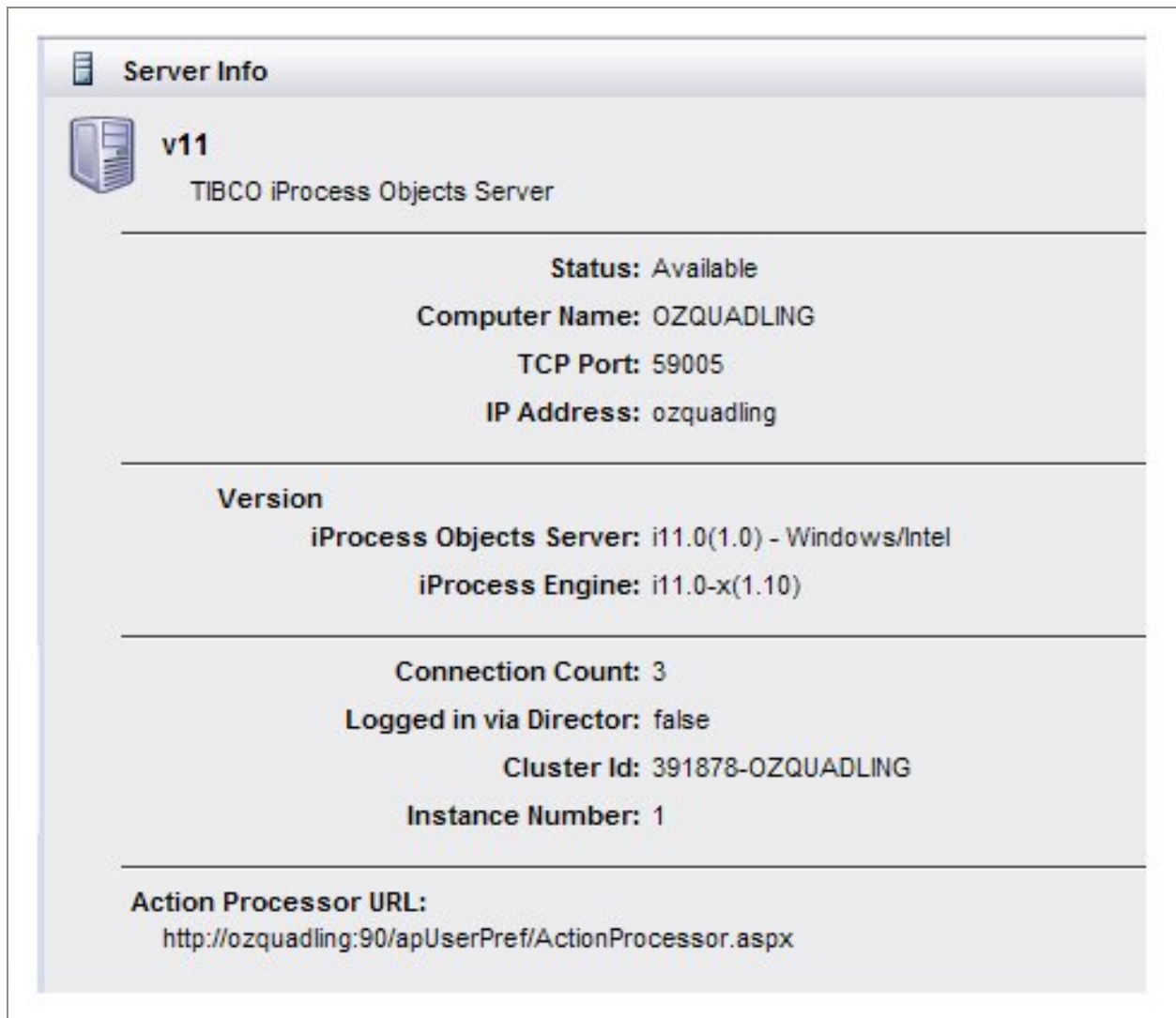
Example

```
this.getApp().ipcShowProcVersion('i111|HIRING|1|0');
```

ipcShowServerInfo

This method displays technical information about the iProcess Objects Server the user is currently logged into.

This method causes a dialog similar to the following to be displayed:



This method is equivalent to clicking on the **Server Info** button in the iProcess Client application.

Syntax

```
this.getApp().ipcShowServerInfo();
```

Parameters

None

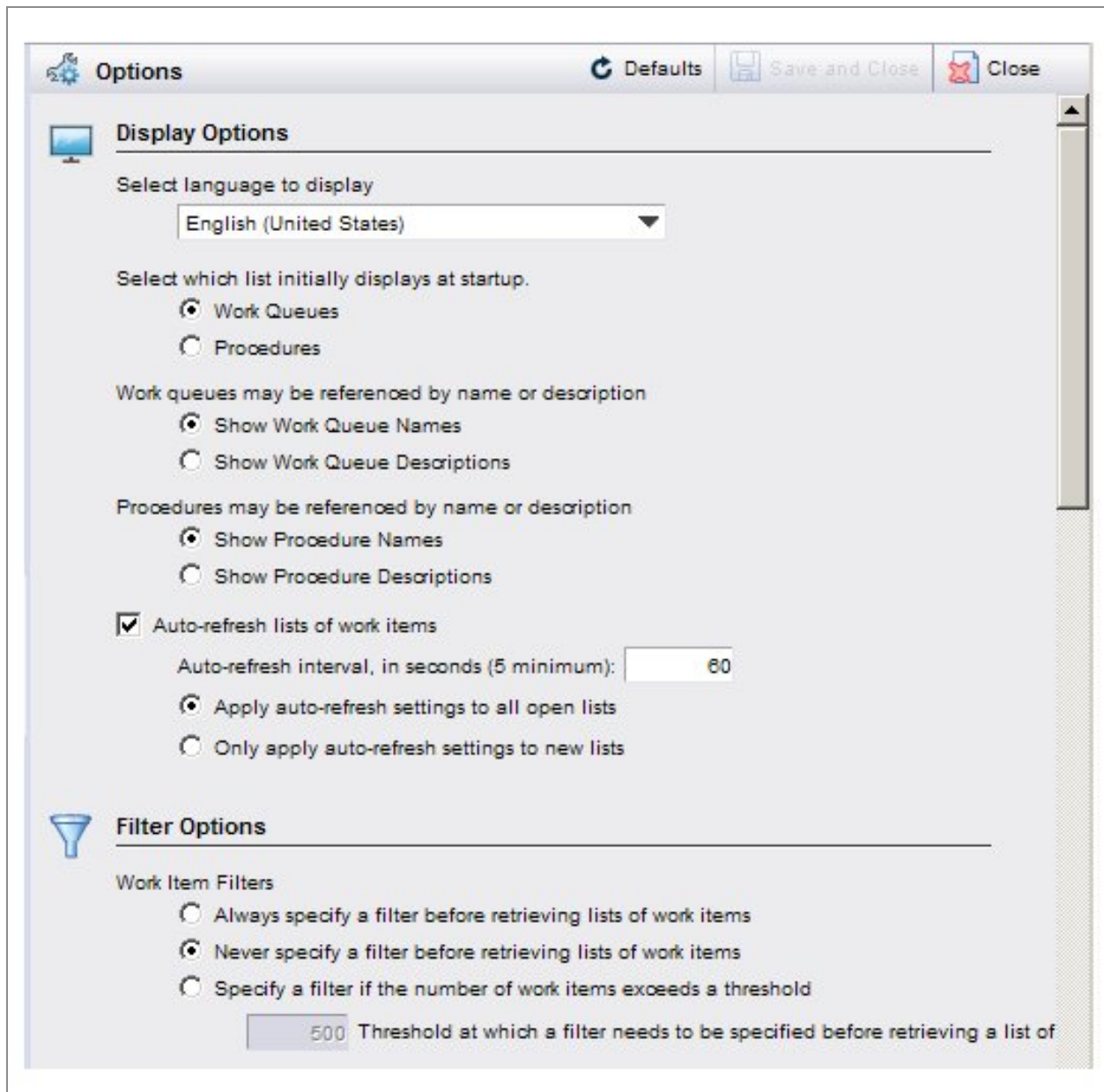
Example

```
this.getApp().ipcShowServerInfo();
```

ipcShowOptions

This method opens the **Options** dialog, which is used to establish default application settings for the user. These include things such as whether preview is turned on by default, the size/location of work item forms, etc.

The **Options** dialog appears as follows:



This method is equivalent to clicking on the **Options** button in the iProcess Client application.

For details about all of the options available, see the *TIBCO iProcess Workspace (Browser) User's Guide*.

i
Note

The only session activity option available with the **ipcShowOptions** method is the change password function. The Session Activity Log is not available using this method.

Syntax

```
this.getApp().ipcShowOptions();
```

Parameters

None

Example

```
this.getApp().ipcShowOptions();
```

ipcWorkItemTag2CaseTag

This method returns a case tag extracted from the work item tag passed as an argument.

If the argument is not a valid work item tag (which includes not containing the correct number of elements between the vertical bars), an exception is thrown.

Syntax

```
this.getApp().ipcWorkItemTag2CaseTag(workItemTag);
```

Parameters

workItemTag - A valid work item tag. For information about tags, see [Introduction](#).

Example

```
var caseTag = this.getApp().ipcWorkItemTag2CaseTag  
('i11|s1|smith|R|102|811|i11|STEP1|0|0');
```

ipcWorkItemTag2WorkQTag

This method returns a work queue tag extracted from the work item tag passed as an argument.

If the argument is not a valid work item tag (which includes not containing the correct number of elements between the vertical bars), an exception is thrown.

Syntax

```
this.getApp().ipcWorkItemTag2WorkQTag(workItemTag);
```

Parameters

workItemTag - A valid work item tag. For information about tags, see [Introduction](#).

Example

```
var workQTag = this.getApp().ipcWorkItemTag2WorkQTag  
( 'i11|s1|smith|R|10|81|i11|s1|0|0' );
```

ipcGetUserAttributes

This method returns an array of objects that represent iProcess attributes assigned to a specific user.

Syntax

```
this.getApp().ipcGetUserAttributes(userName);
```

Parameters

userName - (String) Name of user for which attributes are to be retrieved.

Example

```
var attributes = this.getApp().ipcGetUserAttributes('swadmin');
for (var i = 0; i < attributes.length; i++) {
    var name = attributes[i].Name;
    var type = attributes[i].Type;
    var value = attributes[i].Value;
};
```

ipcGetGroupAttributes

This method returns an array of objects that represent iProcess attributes assigned to a specific group.

Syntax

```
this.getApp().ipcGetGroupAttributes(groupName);
```

Parameters

groupName - (String) Name of group for which attributes are to be retrieved.

Example

```
var attributes = this.getApp().ipcGetGroupAttributes('Supervisors');
for (var i = 0; i < attributes.length; i++) {
    var name = attributes[i].Name;
    var type = attributes[i].Type;
    var value = attributes[i].Value;
};
```

ipcManageMER

This method displays the interface for managing Monitor Event Request (MER) messages on the server. For information about using the Manage MER interface, see [Managing Monitor Event Request \(MER\) Messages](#).

Syntax

```
this.getApp().ipcManageMER();
```

Parameters

None

Example

```
this.getApp().ipcManageMER();
```

IPC Tools Methods Sample

A sample custom GI form is provided that demonstrates how the IPC tools methods can be invoked from a custom GI form.

The sample custom GI form includes, a button for each of the IPC tools methods. The names on the buttons correspond to the method names:

Field Name	Field Type	Field Value
TITLE	swText	
TEXT5	swText	
TEXT4	swText	
TEXT3	swText	

ipcStartCase	procTag	
ipcShowCase	caseTag	☐ hideSummary ☐ hideHistory ☐ hideOutstanding ☐ hideData
ipcCloseCases	caseTags	☐ suppressConfirm (separated by commas)
ipcPurgeCases	caseTags	☐ suppressConfirm (separated by commas)
ipcSuspendCases	caseTags	☐ suppressConfirm (separated by commas)
ipcActivateCases	caseTags	☐ suppressConfirm (separated by commas)
ipcShowGraphicalCaseHistory	caseTag	

Form Details Keep Release Cancel

After setting up the sample (which is described below), you can open the `ipcToolsMethodsForm.js` file in the following directory to see how the methods were implemented on the custom form shown above:

```
InstallDir\JSXAPPS\ipc\components\Forms\ipcToolsMethodsForm\
```

where *InstallDir* is the directory specified during the installation of TIBCO iProcess Workspace (Browser).

Note that this sample requires that you have a procedure deployed on your server. It can be any procedure; you will be using it to display the custom GI form shown above, for the purpose of demonstrating the IPC tools methods.

To set up and use the IPC tools methods sample:

1. Copy the following directory:

```
InstallDir\Samples\ipcToolsMethodsForm\
```

And paste it into the following directory:

```
InstallDir\JSXAPPS\ipc\components\Forms\
```

where *InstallDir* is the directory specified during the installation of TIBCO iProcess Workspace (Browser).

2. Open the iProcess Client's `config.xml` file, which is located in the following directory:

```
InstallDir\JSXAPPS\ipc\
```

3. Locate the **jsxid="Forms"** record in the `config.xml` file, and add a **<Forms>** element for the custom GI form in the sample. An example is shown below:

```
<record jsxid="Forms" type="ipc">
  <Forms>
    <form procName="ALLOCATE" stepName="STEP1"
      class="com.tibco.bpm.ipc.ipcToolsMethodsForm"
      prototypePath="ipcToolsMethodsForm/prototypes/ipcToolsMethodsFormDefault.xml"
      nodeName="DialogTest" floatWorkItems="Dialog"/>
  </Forms>
</record>
```

The attributes in the **<Forms>** element are described below (note that optional attributes can be either omitted or set to a zero-length string).

- **procName** (required) - The name of the procedure in which the custom GI form provided in the sample will be displayed.
- **stepName** (required) - The name of the step on which you want the custom GI form provided in the sample to be displayed. Likely, this will be the first step in the procedure so that the custom form is displayed when a case of the procedure is started.
- **class** (required) - The name of the class that defines the custom form. This is the sample **ipcToolsMethodsForm** class.
- **prototypePath** (optional) - The path to the prototype for the custom form, relative to the forms root directory (which defaults to `'JSXAPPS/ipc/components/Forms/'`).
- **nodeName** (optional) - The name of the TIBCO iProcess Objects Server on which the procedure specified in the **procName** attribute (see above) is defined. This attribute is optional. If it is not specified, any server will match.
- **major** (optional) - The “major” portion of the procedure version number. For example, if the procedure version is 3.2, major = “3”. The default is the most recent.

- **minor** (optional) - The “minor” portion of the procedure version number. For example, if the procedure version is 3.2, minor = “2”. The default is the most recent.
- **floatWorkItems** (optional) - This attribute specifies whether the custom GI form is opened in a separate browser window or a dialog.

The valid entries are “browser” to open the form in a separate browser window, or “dialog” to open the form in a dialog within the browser running the application. If omitted, or set to any other value, the value selected in the "When opening a floating work item form, open it in" option on the application's

Options dialog is used.

- If there was already a custom GI form specified in the `config.xml` file for the procedure and step you added in [Locate the jsxid="Forms" record in the config.xml file, and add a <Forms> element for the custom GI form in the sample. An example is shown below:](#), comment out the original one until you are done using this sample. Once you are done with the sample, you can uncomment the original, then remove the **<Form>** element you added in [Locate the jsxid="Forms" record in the config.xml file, and add a <Forms> element for the custom GI form in the sample. An example is shown below:](#).
- Locate the mapping records in the `config.xml` file (search for '**type="map"**'), and add the following new record, which maps to the **ipcToolsMethodsForm** class:

```
<record jsxid="5" type="map">
  <record jsxid="id" type="string">ipcToolsMethodsForm</record>
  <record jsxid="type" type="string">script</record>
  <record jsxid="load" type="number">1</record>
  <record jsxid="src"
    type="string">JSXAPS/ipc/components/Forms/ipcToolsMethodsForm.js/ipcToolsMethodsFo
rm.js</record>
```

Note that the **jsxid** value for the map-type record (5 in this example) is arbitrary, that is, you can specify any value desired (the **jsxid** is not used in this context).

- Start the iProcess Client, then start a case of the procedure you specified in the **<Form>** element in [Locate the jsxid="Forms" record in the config.xml file, and add a <Forms> element for the custom GI form in the sample. An example is shown below:](#). If you specified the name of the first step in the **stepName** attribute, the custom form showing the IPC tools methods is displayed; if you specified a different step, you will need to progress the case to get to the step you specified.

Most of the IPC tools methods expect a *tag* of some sort (e.g., case tag, work item tag, etc.). Tags are intentionally opaque, that is, we do not provide the information needed to build them — you are expected to acquire them in one of the following ways:

- Tags can be acquired through the iProcess Server Objects object model, specifically using the **getTag** and **makeTag** methods. For information, see the *TIBCO iProcess Server Objects (Java or .NET) Programmer's Guide*.
- You can also acquire tags through an Action Processor response XML. For information, see the *iProcess Workspace (Browser) Action Processor Reference*.

For additional information about GI Forms, see the *GI Forms Interface* section in the *TIBCO iProcess Workspace (Browser) Configuration and Customization* guide.

Forms

This section provides an overview of how forms are handled by the TIBCO iProcess Workspace (Browser).

Introduction to Forms

The iProcess Workspace (Browser) can display the following types of forms:

- TIBCO® General Interface (GI) Forms
- TIBCO Forms
- ASP Forms
- JSP Forms
- BusinessWorks™ FormBuilder Forms
- iProcess Modeler Forms

When a user starts a case that causes a form to display (i.e., when the first step in the procedure is addressed to SW_STARTER), or opens a work item, the iProcess Workspace (Browser) determines which type of form to display by going through the steps listed below.

Note that the order in which it looks for each form type is significant. In other words, if a GI Form is specified for the step, that form type takes precedence; if there is not a GI Form but there is a TIBCO Form specified, that takes precedence over the other form types, and so on.

1. Has a TIBCO® General Interface (GI) form been defined for the step?
 - If the **<Forms>** element in the application's `config.xml` file specifies a GI form for the step/work item, that GI form is displayed.
For more information, see [TIBCO® General Interface Forms Interface](#).
2. Has a TIBCO Form been defined for the step?
 - If a TIBCO Form for the step/work item has been defined and deployed in Business Studio, that TIBCO Form is displayed.

- If you are using TIBCO Forms, the “base” URL of the form’s location must be specified in the **webDAVRoot** parameter in the application’s config.xml file.
 - For more information about the **webDavRoot** parameter, see [WebDAV Root Setting](#).
 - For more information about creating TIBCO Forms, see the *TIBCO Business Studio™ Forms User’s Guide*.
3. Has an ASP Form, JSP Form, or FormBuilder Form been defined for the form?
- One of these form types has been specified for the step if the **ExternalFormURI** parameter in the Action Processors’s configuration file, apConfig.xml, contains a value. This specifies the “base” URL to the form’s location. The remainder of the URL is obtained from the value in the **Formflow Form** field in the step definition.
 - For more information about the **ExternalFormURI** parameter, see [External Form URI](#).
 - For more information about ASP Forms, see [ASP Forms](#).
 - For more information about JSP Forms, see [JSP Forms](#).
 - For more information about FormBuilder Forms, see the *TIBCO BusinessWorks iProcess Forms Plug-in User’s Guide*.
4. A TIBCO iProcess Modeler Form is displayed.
- If none of the previously listed forms are specified for the step, it is assumed the form to display is an iProcess Modeler-created form. If this is the case, the form to display is specified in the step’s definition.
 - For more information, see [Customizing iProcess Modeler Forms](#).

External Forms / GI Forms

Forms are divided into two groups:

- **External Forms** - This category includes the following types of forms:
 - ASP Forms
 - JSP Forms
 - BusinessWorks FormBuilder Forms

- iProcess Modeler Forms
- **GI Forms** - This category includes the following types of forms:
 - TIBCO® General Interface Forms
 - TIBCO Forms

You may see references to external forms and GI Forms in the documentation. Depending on the situation, the system may behave differently for each of these form categories. For instance, you can customize the appearance of the window when displaying work item forms — this is done with the **<BrowserFeatures>** record in the `config.xml` file. The **<BrowserFeatures>** record contains subordinate **<ExternalForms>** and **<GIForms>** records — these subordinate records allow customization for each of the form categories. For more information about setting browser features, see [Specifying Browser Window Features](#).

Also note that when external work item forms are opened, the work item is locked by an external process. Therefore, the work item list does not reflect the locked state of the work item because it was performed externally. Because of this, an **externalLock** configuration parameter is provided that allows you to specify a time delay (in milliseconds) that causes the work item list to be refreshed after a work item is opened externally. The delay provides sufficient time for the external lock of the work item to take place before the list is updated to reflect the locked state. For more information, see [Refresh Work Item List After Work Item is Opened Externally](#).

Dialog/Window Characteristics

When a WCC or client application displays a work item form, it displays it either in a preview pane, in a separate dialog, or in a separate browser window. You can choose which of these formats you want from within the application (for more information, see the *TIBCO iProcess Workspace (Browser) User's Guide*).

Note, however, the type of form you are using determines which of the form formats (preview pane, dialog, or separate browser window) are selectable from the application, as follows:

- if your application uses GI forms, you can choose to open them in any of the three available formats: Preview Pane, dialog, or separate browser window.
- if your application uses external forms, they will always be opened in a separate browser window.

Also note that “dialogs” are further subdivided into the following:

- Webpage dialogs
- Application dialogs

Whether the work item form opens in a “Webpage” dialog or an “application” dialog depends on the setting of the “dialog” attribute in the <**BrowserFeatures**> record in the application’s config.xml file. For more information, see the “dialog” attribute description on [dialog](#).

The following describes the differences in behavior between the different types of dialogs/windows:

- **Minimize/Maximize Buttons** - Webpage dialogs do not have minimize nor maximize buttons. Separate browser windows and application dialogs have these buttons.
- **Floating Window Outside Application Window** - Both Webpage dialogs and separate browser windows can be floated outside the parent application's window, whereas application dialogs cannot.
- **Browser Feature Attributes** - The Browser Feature attributes (i.e., the attributes of the <**BrowserFeatures**> record in the config.xml file) supported depends on the dialog/window and the type of browser used, as follows:
 - **Webpage dialog:** If using Internet Explorer, only the "resizable" and "status" attributes are supported. If using Firefox, the supported attributes are: "dialog", "directories", "location", "menubar", "minimizable", and "toolbar".
 - **Application dialog:** None of the Browser Feature attributes are supported for this type of dialog.
 - **Separate browser window:** The table on [Browser Feature Attributes](#) lists the browser features that are supported for each of the available browsers.
- **Close as child window:** Both Webpage dialogs and application dialogs are children of the parent window, therefore if the parent window is closed (or minimized), the Webpage/application dialog is also closed (or minimized). Separate browser windows do not close (or minimize) when the parent is closed (or minimized).

TIBCO® General Interface Forms Interface

This section provides an overview of the TIBCO GI Forms interface, information about how to implement the interface to display custom GI forms, and a list of the methods available to handle custom form prototypes.

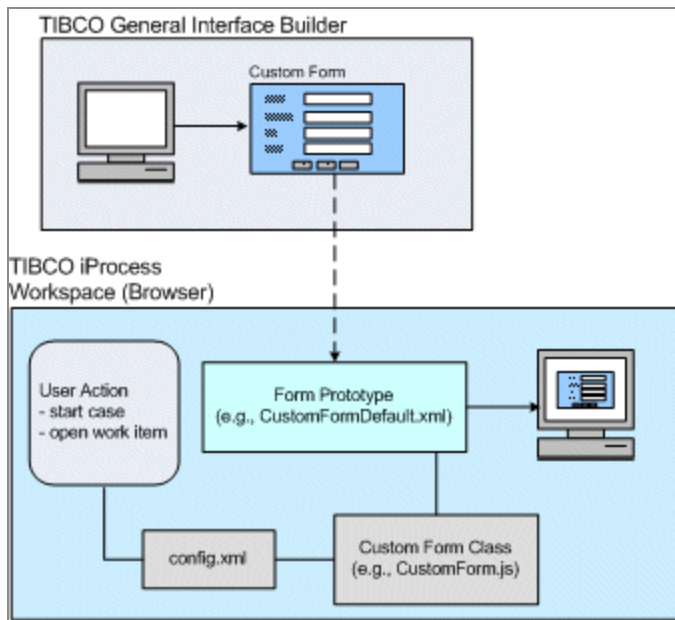
The intended audience of this information is developers who have a thorough understanding of the TIBCO® General Interface Builder and the functionality available through the TIBCO iProcess Server Objects.

Overview

The GI Forms interface allows you to create forms using TIBCO® General Interface Builder, then use those forms for work item steps in the iProcess Workspace (Browser). This allows you to take advantage of the advanced form-building capabilities of the TIBCO® General Interface Builder.

The GI Forms interface allows your custom forms to utilize the existing iProcess Workspace (Browser) communication methods to perform the following functions:

- start a case
- lock a work item
- get field data to display in the form
- keep or release the work item



When a user either starts a case of a procedure or opens an existing work item, the TIBCO iProcess Workspace (Browser) will check to see if there is a GI form specified for that step in the **<Forms>** element of the client's `config.xml` file:

- If there is a GI form specified for the step:
 - The iProcess Workspace (Browser) instantiates the custom GI form class that is specified in the **<Forms>** element in the iProcess Workspace (Browser) configuration file (`config.xml`). This is described in more detail later.
 - Field data is requested from the Action Processor (applicable only when opening a work item, not when starting a case).
 - The custom GI form (prototype) is displayed within the GI context in the iProcess Workspace (Browser).
 - A keep or release request is sent to the Action Processor when the user initiates one of those actions.
- If there is no GI form definition for the step:
 - The normal iProcess Workspace (Browser) open work item actions are performed, i.e., a new browser window is opened pointing to the Action Processor URL to handle the form action. This delegates control to the external form handling process.

**Note**

For information about the version of TIBCO® General Interface Builder that must be used if you are developing your own GI forms to use with the iProcess Workspace (Browser), see the Release Notes for the iProcess Workspace (Browser).

Base Class

The GI Forms interface provides the following base class that is extended for each custom GI form created:

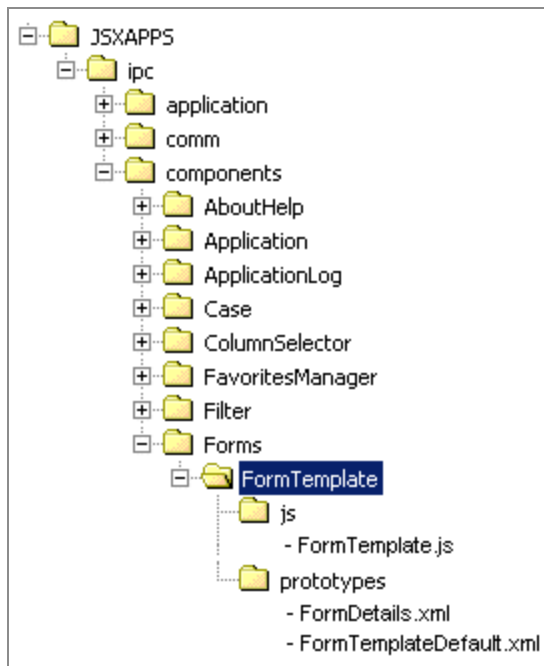
- `com.tibco.bpm.ipc.Form`

Each custom form consists of two files:

- `<FormClassName>.js` - This class extends the **`com.tibco.bpm.ipc.Form`** base class, adding any form-specific logic.
- `<FormClassName>.xml` - This defines the GUI prototype for the form component layout.

Sample Implementation

A sample subclass implementation of the GI Forms interface base class is provided in the **FormTemplate** subdirectory.



The **FormTemplate.js** file is a sample implementation class (**com.tibco.bpm.ipc.FormTemplate**), which you can use by replacing “FormTemplate” with your custom form class name, then modifying the methods in the class to fit your custom form needs. The **com.tibco.bpm.ipc.FormTemplate** class extends the base class (**com.tibco.bpm.ipc.Form**). For information about the properties and methods available in the base class, see [Interface Properties and Methods](#). (Note that at a minimum, the custom class that extends the **com.tibco.bpm.ipc.Form** base class must override the **postLoadInit**, **doKeep**, and **doRelease** methods; it can also optionally override the **init** and **doCancel** methods.)

Multiple prototypes can be defined for each custom form. During the implementation phase, you will specify which prototype to use. (The prototype can also be specified using the **prototypePath** attribute of the **<Form>** element in the **config.xml** file.)



Note

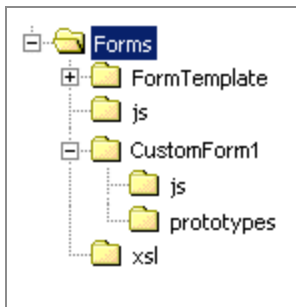
The sample implementation also contains a **showFormDetails** function (which is called by the **Form Details** button on the sample template). This function provides details about the messages sent to and returned by the Action Processor, which may be useful during development. It uses the **FormDetails.xml** prototype.

Implementation

Perform the following steps to implement the GI Forms interface in the TIBCO iProcess Workspace (Browser):

1. Copy the entire FormTemplate directory, and paste it into the .../components/Forms directory.
2. Rename the copied directory to match the class name for your custom form.

For example, assume a custom form with a class name of “CustomForm1”:



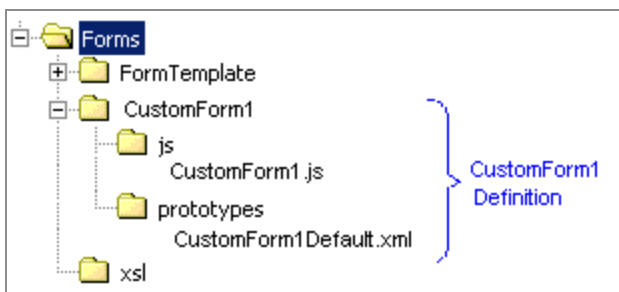
3. In the new custom form directory, replace the “FormTemplate” name with the name of your custom form class. For example:

- Forms/CustomForm1/js/**FormTemplate.js**
- Forms/CustomForm1/prototype/**FormTemplateDefault.xml**

... should become:

- Forms/CustomForm1/js/**CustomForm1.js**
- Forms/CustomForm1/prototype/**CustomForm1Default.xml**

The following illustrates an example directory/file structure for the new custom form:



4. Globally replace “FormTemplate” with your class name (“CustomForm1” in this example) in the CustomForm1.js file.

5. Replace the content of CustomForm1Default.xml with the XML that defines the prototype of your custom form.

Event actions defined in a custom form prototype can get reference to the custom form class instance by getting the parent of the top level object. The examples below are taken directly from the FormTemplateDefault.xml sample prototype file.

The example button object defined below returns a reference to the custom form class instance, and the **doKeep()** function defined in the custom class is called when the **jsxexecute** event is triggered.

```
<object type="jsx3.gui.Button">
  <variants jsindex="0" jsheight="18"/>
  <strings jsname="btnKeep" jsxtext="Keep" jsmargin="top:0px"/>
  <events jsxexecute="this.getAncestorOfName('layoutFormData').getParent().doKeep();"/>
```

Note that the name referenced above, **layoutFormData**, is the name assigned to the top-level object (using: jsxname="layoutFormData") in the form prototype:

```
<object type="jsx3.gui.LayoutGrid">
  <variants jsrepeat="2" jsisarray="T"/> <jsrelativeposition="0" />
  <strings jsname="layoutFormData" jswidth="100%" jsheight="100%" />
  <children>
```

See JSXAPPS\ipc\components\Forms\FormTemplate\prototypes\FormTemplateDefault.xml to see these examples in their complete context.

6. Modify the methods in CustomForm1.js to handle the custom form prototype. For information about the methods available, see [Base Class Methods](#).
7. Update the TIBCO iProcess Workspace (Browser) configuration file (... \JSXAPPS\ipc\config.xml) to include a **<Form>** element under the **<Forms>** element for each custom form.

The following is an example **<Forms>** element with a number of sample **<Form>** entries:

```

<record jsxid="Forms" type="ipc">
  <Forms>
    <Form
      procName="ALLOCATE"
      stepName="ASSIGN"
      class="com.tibco.bpm.ipc.FormTemplate"
      prototypePath = "FormTemplate/prototypes/FormTemplateDefault.xml"
      nodeName="nodeName1"
      major="0"
      minor="2"
      floatWorkItems="browser"/>
    <Form
      procName="ALLOCATE"
      stepName="SUMMARY"
      class="com.tibco.bpm.ipc.FormTemplate"
      prototypePath = ""
      nodeName=""
      major=""
      minor=""
      floatWorkItems="dialog"/>
    <Form
      procName="ORDER"
      stepName="REQUEST"
      class="com.tibco.bpm.ipc.FormTemplate"
      floatWorkItems=""/>
    <Form
      procName="ORDER"
      stepName="STOCK"
      class="com.tibco.bpm.ipc.FormTemplate"/>
  </Forms>
</record>

```



Note

A sample configuration file (config-sample.xml) is provided in the JSXAPPS/ipc/components/Forms directory from which you can copy a sample **<Forms>** element, then modify it to fit your needs.

The following defines the available **<Form>** element attributes that need to be specified for each custom form (note that the optional attributes can be either omitted or set to a zero-length string):

- **procName** (required) - The name of the procedure in which the custom form is used.
- **stepName** (required) - The name of the step to which the custom form is associated. When this step is reached in the case (procedure instance), the custom form is displayed.

- **class** (required) - The name of the class that defines the custom form. This is the class that is instantiated by the iProcess Workspace (Browser) when the user starts a case or opens an existing work item. For the example described earlier, this would be specified as:

```
class="com.tibco.bpm.ipc.CustomForm1"
```

- **prototypePath** (optional) - The path to a prototype for the custom form, relative to the forms root directory, specified in **com.tibco.bpm.ipc.Form.DIR** (this defaults to 'JSXAPPS/ipc/components/Forms/'). For the example described earlier, this would be specified as:

```
prototypePath="CustomForm1/prototypes/CustomForm1Default.xml"
```

If the prototype path is not specified here or explicitly in the class, the default path is used:

```
com.tibco.bpm.ipc.Form.DIR + classname + '/prototypes/' +  
  classname + 'Default.xml'
```

For example, the default prototype path for the sample FormTemplate is:

```
"FormTemplate/prototypes/FormTemplateDefault.xml"
```

- **nodeName** (optional) - The name of the TIBCO iProcess Objects Server on which the procedure specified in the **procName** attribute (see above) is defined. This attribute is optional. If it is not specified, any server will match.
 - **major** (optional) - The “major” portion of the procedure version number. For example, if the procedure version is 3.2, major = “3”.
 - **minor** (optional) - The “minor” portion of the procedure version number. For example, if the procedure version is 3.2, minor = “2”.
 - **floatWorkItems** (optional) - This attribute is relevant only for custom GI forms. It specifies whether or not the custom GI form is opened in a separate browser window. The valid entries are “browser” for open the form in a separate browser window, or “dialog” for open the form in a dialog within the browser running the TIBCO iProcess Workspace (Browser). If omitted or set to any other value, the value selected in the **When opening a floating work item form, open it in** option on the application’s **Options** dialog is used.
8. Update the TIBCO iProcess Workspace (Browser) configuration file to include mappings to each custom form class. The configuration file is located at:

...\JSXAPPS\ipc\config.xml

The following is an example map record for the sample form template:

```
<record jsxid="127" type="map">
  <record jsxid="id" type="string">FormTemplate</record>
  <record jsxid="type" type="string">script</record>
  <record jsxid="owner" type="string">application</record>
  <record jsxid="onLoad" type="boolean">true</record>
  <record jsxid="required" type="boolean">true</record>
</record>
```

Locate the existing mapping records (type = “map”) in the config.xml file and enter a record for each of your custom form classes.

Note that the **jsxid** value for the map-type record (127 in this example) is arbitrary, i.e., you can specify any value desired (the **jsxid** is not used in this context).



Note

A sample configuration file (config-sample.xml) is provided in the JSXAPPS/ipc/components/Forms directory from which you can copy sample mapping records, then modify them to fit your needs.

Interface Properties and Methods

This section provides information about the properties and methods available in the **com.tibco.bpm.ipc.Form** base class. Since the custom form class extends this base class, these properties and methods can be accessed directly in your custom form class.

Base Class Properties

The following accessor methods are available to the subclass to access properties in the **com.tibco.bpm.ipc.Form** base class.

Method	Returns	Property Description
isStartCase()	boolean	True if form opened for a start case.
getDescription()	string	Start case or work item description.

Method	Returns	Property Description
getProcName()	string	The procedure name.
getStepName()	string	Name of the start step.
getProcTag()	string	The procedure tag.
getCaseNumber()	string	The case number. ¹
getWorkItemTag()	string	The work item tag. ^a
getWorkQTag()	string	The work queue tag. ^a
getNode()	jsx3.xml.Entity	The start case or work item node.

Note that any of the values in the node returned by **getNode()** can be read using **getAttribute('attributeName')** as shown below:

```
var version = this.getNode().getAttribute('Version');
```

The following samples show node data returned by **getNode()** (shown formatted):

For Start Case:

```
<record
  jsxid="IDAOAICE"
  jsxttext=""
  jsximg=""
  IsStatus="true"
  IsStatusImage="JSXAPPS/ipc/application/images/ProcReleased.gif"
  Name="PROCNAME"
  Description="Case Description"
  HostingNode="serverNodeName"
  Version="0.2"
  Tag="serverNodeName | PROCNAME | 0 | 2"
  ProcNumber="36"
  StartStepName="STEP1"
  Status="swReleased"
  CaseDescOpt="swRequiredDesc"
  IsAutoPurge="false"
```

¹The case number, work item tag, and work queue tag are null for start case.

```

IsIgnoreBlank="false"
IsNetworked="false"
IsSubProc="false"
IsSubProcImage="JSXAPPS/ipc/application/images/IsFalse.gif"
IsOrphaned="false"
IsWorkDays="true"
IsPrediction="false"
Owner="username"
Duration="swDurationNone"
Permission="Start / History"
CaseCount="36"
ActiveCount="35"
ClosedCount="1"
jsxselected="1"
ListId="_jsx_ipcNS_102"
IsCustomFormStartCase="true">
</record>

```

For Work Item:

```

<record
  jsxid="IDAFSZGE"
  jsxttext=""
  jsximg=""
  IsStatus="true"
  IsStatusImage="JSXAPPS/ipc/application/images/ItemLockedDesktop.gif"
  CaseNumber="4922"
  CaseReference="36-4922"
  Description="Case Description"
  Tag="serverNodeName | PROCNAME | username | R | 4922 | 421916 |
                                           serverNodeName | STEP1 | 0 | 2"
  StartedBy="username@serverNodeName"
  Proc_Name="PROCNAME"
  Proc_Description="Proc Description"
  Proc_HostingNode="serverNodeName"
  Version="0.2"
  Proc_Tag="serverNodeName | PROCNAME | 0 | 2"
  ComputerName="OZQUADLING"
  CaseFields=""
  MailId="421916 | serverNodeName"
  CaseTag="serverNodeName | PROCNAME | 0 | 2 | 4922"
  AddrToName="username@serverNodeName"
  Arrived="2006-08-29 16:22"
  IsDeadline="false"
  IsDeadlineImage="JSXAPPS/ipc/application/images/IsFalse.gif"
  IsDeadlineExp="false"
  IsDeadlineExpImage="JSXAPPS/ipc/application/images/IsFalse.gif"

```

```

IsKeepOnWithdrawalImage="JSXAPPS/ipc/application/images/IsFalse.gif"
IsKeepOnWithdrawal="false"
IsForwardableImage="JSXAPPS/ipc/application/images/IsFalse.gif"
IsForwardable="false"
IsLocked="false"
IsLockedImage="JSXAPPS/ipc/application/images/IsFalse.gif"
IsLongLocked="true"
IsLongLockedImage="JSXAPPS/ipc/application/images/IsTrue.gif"
IsOrphanedImage="JSXAPPS/ipc/application/images/IsFalse.gif"
IsOrphaned="false"
IsReleasable="false"
IsReleasableImage="JSXAPPS/ipc/application/images/IsFalse.gif"
IsUnopened="false"
IsUnopenedImage="JSXAPPS/ipc/application/images/IsFalse.gif"
IsUrgent="false"
IsUrgentImage="JSXAPPS/ipc/application/images/IsFalse.gif"
IsSuspended="false"
IsSuspendedImage="JSXAPPS/ipc/application/images/IsFalse.gif"
LockedBy="username"
Priority="50"
StepName="STEP1"
StepDescription="First step"
OCCUPATION=""
WorkQParam1=""
WorkQParam2=""
WorkQParam3=""
WorkQParam4=""
DeltaStatus="swNotDeltaItem"
jsxselected="1"
ListId="_jsx_ipcNS_254"
IsCustomFormStartCase="false">
</record>

```

Base Class Methods

This section provides information about the methods available in the **com.tibco.bpm.ipc.Form** base class.

Note that at a minimum, the custom class that extends the **com.tibco.bpm.ipc.Form** base class must override the **postLoadInit**, **doKeep**, and **doRelease** methods; it can also optionally override the **init** and **doCancel** methods.

The following is a list of the methods described in this section:

- [buildCDFArrays](#)

- [closeForm](#)
- [confirmUserMessage](#)
- [createFieldDefsRequest](#)
- [createKeepRequest](#)
- [createLockRequest](#)
- [createReleaseRequest](#)
- [doCancel](#)
- [doClose](#)
- [doKeep](#)
- [doRelease](#)
- [getWindowContext](#)
- [init](#)
- [lockWorkItem](#)
- [onBeforeUnload](#)
- [postLoadInit](#)
- [readFieldDefs](#)
- [readFormFields](#)
- [readStepMarkings](#)
- [showUserMessage](#)
- [socketRequest](#)
- [transformData](#)

buildCDFArrays

Purpose

This method examines the cached XML CDF document, created by the **lockWorkItem** method, and identifies any array field elements (the **ssoFieldType** attribute will start with “swArrayOf” for array fields).

Each array field record contains the field name and a value string containing the values of all array elements, separated by a “|” character (e.g., “a|b|c|d”).

A new CDF record is created for each array element. The name of the array element includes the array index (e.g., “field[0]”) and its value is extracted from the string containing all element values.

Syntax

```
buildCDFArrays()
```

Parameters

None

Returns

CDF records

Remarks

This method is called by the **initializeFormData** function to handle array field data.

closeForm

Purpose

This method closes the dialog.

Syntax

```
closeForm()
```

Parameters

None

Returns

Void

Remarks

This method also clears the XML cache data. The XML cache data includes any XML data stored in cache as a result of a **socketRequest** or **transformData** call. Caches are cleared that have IDs containing the form instance object ID. If names other than the default cache names have been used, the overriding class should ensure that all caches have been removed.

confirmUserMessage

Purpose

Displays a confirm dialog with the given message. If this is a child browser window, the confirm is sent to the child window context to prevent focus being moved to the parent browser window.

Note - Custom GI forms should call **this.confirmUserMessage** (vs. confirm) to prevent window focus from shifting back to the parent window.

Syntax

```
confirmUserMessage(message)
```

Parameters

Parameter	Type	Required?	Description
message	string	Yes	The message to be displayed in the confirm dialog.

Returns

Boolean — True if the user selects **OK** from the confirm dialog; False if the user selects **Cancel**.

createFieldDefsRequest

Purpose

This method creates and returns the Action Processor **GetFieldDefs** request, which is used to get the field definitions for a procedure for which you are starting a case or opening a work item.

This method is used if you don't have knowledge of the fields on the form. The **GetFieldDefs** request returns all fields defined for the procedure.

This method only returns the **GetFieldDefs** request; it does not submit it to the Action Processor. To submit the request, you must call the **socketRequest** method (see [socketRequest](#)).

Syntax

```
createFieldDefsRequest(requestId)
```


Parameters

Parameter	Type	Required?	Description
requestId	string	No	Identifies the request. If not provided, the xmlCacheldAp is used (which can be obtained with the this.getXmlCacheldAp function).

Returns

A string — **GetFieldDefs** request XML.

Remarks

This method is called by **readFieldDefs**. Normally, a developer of a custom form will have knowledge of the fields that are available on the form. The sample file, **com.tibco.bpm.ipc.FormTemplate**, can be applied to any procedure; it calls **readFieldDefs** to read the fields that are available.

This method is not required for use with a form that has statically defined fields on it.

Example XML

The following provides example XML for this method.

Field Defs Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<ap:Action xmlns:ap="http://tibco.com/bpm/actionprocessor"
  xmlns:sso="http://tibco.com/bpm/sso/types">
  <ap:ProcManager>
    <ap:GetFieldDefs Id="_jsx_ipcNS_1394_XML_apfd">
      <sso:ProcTag>i2tagtest|ALLOCATE|0|2</sso:ProcTag>
    </ap:GetFieldDefs>
  </ap:ProcManager>
</ap:Action>
```

Field Defs Request Result:

```
<ap:ActionResult xmlns:ap="http://tibco.com/bpm/actionprocessor">
  <ap:Status>
    <ap:Version>10.6.0</ap:Version>
    <ap:ReturnCode>0</ap:ReturnCode>
    <ap:ReturnComment>The Action was processed successfully. Check the
individual      Request Results for their status.</ap:ReturnComment>
    <ap:ReturnDateTime>2006-02-06T10:43:42.432-0800</ap:ReturnDateTime>
  </ap:Status>
  <ap:SSO>
    <sso:vSSOData xmlns:sso="http://tibco.com/bpm/sso/types">
      <sso:Results>
        <sso:vResult Id="_jsx_ipcNS_301_XML_apfd">
          <sso:FieldDefs>
            <sso:vFieldDef>
              <sso:Name>LOCATION</sso:Name>
              <sso:FieldType>swText</sso:FieldType>
              <sso:Value></sso:Value>
              <sso:Length>20</sso:Length>
              <sso:DecimalPlaceCnt>0</sso:DecimalPlaceCnt>
              <sso:IsArrayField>>false</sso:IsArrayField>
            </sso:vFieldDef>
            <sso:vFieldDef>
              <sso:Name>MEMO</sso:Name>
              <sso:FieldType>swMemo</sso:FieldType>
              <sso:Value></sso:Value>
              <sso:Length>0</sso:Length>
              <sso:DecimalPlaceCnt>0</sso:DecimalPlaceCnt>
              <sso:IsArrayField>>false</sso:IsArrayField>
            </sso:vFieldDef>
            <sso:vFieldDef>
              <sso:Name>NAME</sso:Name>
              <sso:FieldType>swText</sso:FieldType>
              <sso:Value></sso:Value>
              <sso:Length>20</sso:Length>
              <sso:DecimalPlaceCnt>0</sso:DecimalPlaceCnt>
              <sso:IsArrayField>>false</sso:IsArrayField>
            </sso:vFieldDef>
            <sso:vFieldDef>
              <sso:Name>NUMERIC</sso:Name>
              <sso:FieldType>swNumeric</sso:FieldType>
              <sso:Length>10</sso:Length>
              <sso:DecimalPlaceCnt>2</sso:DecimalPlaceCnt>
              <sso:IsArrayField>>false</sso:IsArrayField>
            </sso:vFieldDef>
            <sso:vFieldDef>
              <sso:Name>REASON</sso:Name>
```

```

        <sso:FieldType>swText</sso:FieldType>
        <sso:Value></sso:Value>
        <sso:Length>20</sso:Length>
        <sso:DecimalPlaceCnt>0</sso:DecimalPlaceCnt>
        <sso:IsArrayField>>false</sso:IsArrayField>
    </sso:vFieldDef>
    <!-- SW fields removed -->
</sso:FieldDefs>
</sso:vResult>
</sso:Results>
</sso:vSSOData>
</ap:SSO>
</ap:ActionResult>

```

Field Defs Request Transform XSL:

```

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:sso="http://tibco.com/bpm/sso/types"
xmlns:ap="http://tibco.com/bpm/actionprocessor" exclude-result-
prefixes="sso ap">
    <xsl:output omit-xml-declaration="yes"/>
    <xsl:param name="uniqueId"/>
    <xsl:output indent="yes"/>
    <xsl:template match="/">
        <data>
            <xsl:apply-templates
                select="/ap:ActionResult/ap:SSO/sso:vSSOData/sso:Results/sso:vRe
sult[@Id = $uniqueId]"/>
        </data>
    </xsl:template>
    <xsl:template match="sso:vResult">
        <xsl:apply-templates select="sso:FieldDefs/sso:vFieldDef"/>
    </xsl:template>
    <xsl:template match="sso:vFieldDef">
        <xsl:variable name="fieldName" select="sso:Name"/>
        <xsl:if test="not(starts-with($fieldName, 'SW_'))">
            <xsl:element name="record">
                <xsl:attribute name="jsxid">
                    <xsl:value-of select="generate-id()"/>
                </xsl:attribute>
                <xsl:attribute name="ssoName">
                    <xsl:value-of select="sso:Name"/>
                </xsl:attribute>
                <xsl:attribute name="ssoFieldType">
                    <xsl:value-of select="sso:FieldType"/>
                </xsl:attribute>
            </xsl:element>
        </xsl:if>
    </xsl:template>

```

```

        </xsl:attribute>
        <xsl:attribute name="ssoValue">
            <xsl:value-of select="sso:Value"/>
        </xsl:attribute>
    </xsl:element>
</xsl:if>
</xsl:template>
</xsl:stylesheet>

```

Field Defs Request CDF:

```

<data>
  <record jsxid="IDARQ4YC" ssoName="LOCATION" ssoFieldType="swText"
ssoValue=""/>
  <record jsxid="IDAYQ4YC" ssoName="MEMO" ssoFieldType="swMemo"
ssoValue=""/>
  <record jsxid="IDA5Q4YC" ssoName="NAME" ssoFieldType="swText"
ssoValue=""/>
  <record jsxid="IDAGR4YC" ssoName="NUMERIC" ssoFieldType="swNumeric"
ssoValue=""/>
  <record jsxid="IDAMR4YC" ssoName="REASON" ssoFieldType="swText"
ssoValue=""/>
  <record jsxid="IDA1T4YC" ssoName="TEXT" ssoFieldType="swText"
ssoValue=""/>
  <record jsxid="IDACU4YC" ssoName="TITLE" ssoFieldType="swText"
ssoValue=""/>
</data>

```

createKeepRequest

Purpose

If a case is being started, this method creates and returns the Action Processor **StartCase** request, with the **isRelease** flag set to False.

If an existing work item is being *kept* in the work queue, this method creates and returns the Action Processor **KeepItems** request.

This method only returns the **StartCase** or **KeepItems** request; it does not submit it to the Action Processor. To submit the request, you must call the **socketRequest** method (see [socketRequest](#)).

Syntax

```
createKeepRequest(fields,
                  requestId,
                  validate,
                  subProcPrecedence)
```

Parameters

Parameter	Type	Required?	Description
fields	Object or Array of Objects	Yes	This can be either: - An Array of Objects containing field data properties: name, fieldType, value. - An Object, such as com.tibco.bpm.ipc.FieldData that implements <code>getFieldDataArray()</code>, which returns an Array (for information about com.tibco.bpm.ipc.FieldData, see FieldData Class).
requestId	string	No	Identifies the request. If not provided, xmlCacheIdAp is used (which can be obtained with the this.getXmlCacheIdAp function).
validate	boolean	No	If True, the field values are validated against the type to which they are defined (text, date, time, etc.). Default = False
subProcPrecedence	string	No	Specifies the precedence of sub-procedure statuses that are launched

Parameter	Type	Required?	Description
			from the procedure. These are enumerated in <code>SWSubProcPrecedenceType</code>. Default = If not specified, the option set in the application is used (Options dialog > General tab > Sub-Case Version Precedence). Note: The <code>subProcPrecedence</code> parameter is only applicable for <code>StartCase</code>.

Returns

A string — **StartCase** or **KeepItems** request XML

Remarks

This method is called by the custom form class **doKeep** method. The **Keep** button on the form triggers the call to the **doKeep** method. Data is collected from the form and the XML request is obtained using this method. This request is then submitted to the Action Processor using the **socketRequest** method.

Example

The following is an example usage from **com.tibco.bpm.ipc.FormTemplate**:

```
ipcClass.prototype.doKeep = function() {
    var keepRequest = this.createKeepRequest(this.getFormData());
    var socket = this.socketRequest(keepRequest);
    if (socket.isSuccess() && socket.getSsoErrorMsg() == null) {
        this.jsxsuper();
    }
};
```

Example XML

The following provides example XML for this method.

Keep Request:

```

<?xml version="1.0" encoding="UTF-8"?>
<ap:Action xmlns:ap="http://tibco.com/bpm/actionprocessor"
  xmlns:sso="http://tibco.com/bpm/sso/types">
  <ap:WorkQ>
    <ap:KeepItems Id="_jsx_ipcNS_1394_XML_ap">
      <sso:WorkQTag>i2tagtest|swadmin|R</sso:WorkQTag>
      <sso:WIFieldData>
        <sso:vWIFieldGroup>
          <sso:WorkItemTag>i2tagtest|ALLOCATE|swadmin|R|2708|414137|
            i2tagtest|STEP1|0|2</sso:WorkItemTag>
          <sso:WorkItemFields>
            <sso:vField>
              <sso:Name>LOCATION</sso:Name>
              <sso:FieldType>swText</sso:FieldType>
              <sso:Value>USA</sso:Value>
            </sso:vField>
            ...
          </sso:WorkItemFields>
        </sso:vWIFieldGroup>
      </sso:WIFieldData>
      <sso:ValidateFields/>
    </ap:KeepItems>
  </ap:WorkQ>
</ap:Action>

```

Keep Request (Start Case):

```
<?xml version="1.0" encoding="UTF-8"?>
<ap:Action xmlns:ap="http://tibco.com/bpm/actionprocessor"
  xmlns:sso="http://tibco.com/bpm/sso/types">
  <ap:User>
    <ap:StartCase Id="_jsx_ipcNS_217_XML_ap">
      <sso:ProcTag>i2tagtest|ALLOCATE|0|2</sso:ProcTag>
      <sso:Description>Demo</sso:Description>
      <sso:ReleaseItem>false</sso:ReleaseItem>
      <sso:ValidateFields>false</sso:ValidateFields>
      <sso:Fields>
        <sso:vField>
          <sso:Name>LOCATION</sso:Name>
          <sso:FieldType>swText</sso:FieldType>
          <sso:Value>USA</sso:Value>
        </sso:vField>
        ...
      </sso:Fields>
    </ap:StartCase>
  </ap:User>
</ap:Action>
```

createLockRequest

Purpose

This method creates and returns the Action Processor **LockItems** request.

This method only returns the **LockItems** request; it does not submit it to the Action Processor. To submit the request, you must call the **socketRequest** method (see [socketRequest](#)).

Note that it is not required that you call this method — it is called by the custom form class **lockWorkItem** method. It is available, however, if you want the Action Processor **LockItems** request to lock a work item and get field data.

Syntax

```
createLockRequest(fieldNames,
  requestId)
```


Parameters

Parameter	Type	Required?	Description
fieldNames	Array of strings	Yes	Fields to include in the lock items request.
requestId	string	No	Identifies the request. If not provided, the xmlCacheIdAp is used (which can be obtained with the this.getXmlCacheIdAp function).

Returns

A string — **LockItems** request XML

Example

The following is an example usage from **com.tibco.bpm.ipc.Form**:

```
ipcClass.prototype.lockWorkItem = function(
    fieldNames, xslPath, apCacheId, cdfCacheId) {
    var fieldData = null;
    var useApCacheId = apCacheId != null ? apCacheId :
        this.getXmlCacheIdAp();
    var useXslPath = xslPath != null ? xslPath :
        com.tibco.bpm.ipc.Form.DIR +
        'xsl/lockItemsToCdf.xsl';
    var useCdfCacheId = cdfCacheId != null ? cdfCacheId :
        this.getXmlCacheId();
    var lockRequest = this.createLockRequest(fieldNames, useApCacheId);
    var socket = this.socketRequest(lockRequest, useApCacheId);
    if (socket.isSuccess() && socket.getSsoErrorMsg() == null) {
        this.transformData(useXslPath, useApCacheId, useCdfCacheId);
        var doc = this.getApp().getCache().getDocument(useCdfCacheId);
        fieldData = new com.tibco.bpm.ipc.FieldData();
        fieldData.loadFromCdfDoc(doc);
    }
    return fieldData;
};
```

Example XML

The following provides example XML for this method.

Lock Items Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<ap:Action xmlns:ap="http://tibco.com/bpm/actionprocessor"
  xmlns:sso="http://tibco.com/bpm/sso/types">
  <ap:WorkQ>
    <ap:LockItems Id="y76R20">
      <sso:WorkQTag>i2tagtest|swadmin|R</sso:WorkQTag>
      <sso:WorkItemTags>
        <sso:string>i2tagtest|ALLOCATE|swadmin|R|2050|408966|i2tagtes
t|
          SUMMARY|0|2</string>
      </sso:WorkItemTags>
      <sso:WIFGContent>
        <sso:WIFieldNames>
          <sso:string>LOCATION</sso:string>
          <sso:string>MEMO</sso:string>
          <sso:string>NAME</sso:string>
          <sso:string>NUMERIC</sso:string>
          <sso:string>REASON</sso:string>
          <sso:string>TEXT</sso:string>
          <sso:string>TITLE</sso:string>
        </sso:WIFieldNames>
        <sso:FieldsOption>ssoAllMarkings</sso:FieldsOption>
      </sso:WIFGContent>
    </ap:LockItems>
  </ap:WorkQ>
</ap:Action>
```

Lock Items Request Result:

```

<ap:ActionResult xmlns:ap="http://tibco.com/bpm/actionprocessor">
  <ap:Status>
    <ap:Version>10.6.0</ap:Version>
    <ap:ReturnCode>0</ap:ReturnCode>
    <ap:ReturnComment>The Action was processed successfully. Check the
individual      Request Results for their status.</ap:ReturnComment>
    <ap:ReturnDateTime>2006-02-06T10:43:42.698-0800</ap:ReturnDateTime>
  </ap:Status>
  <ap:SSO>
    <sso:vSSOData xmlns:sso="http://tibco.com/bpm/sso/types">
      <sso:Results>
        <sso:vResult Id="_jsx_ipcNS_301_XML_ap">
          <sso:WIFieldGroups>
            <sso:vWIFieldGroup>
              <sso:WorkItemTag>i2tagtest|ALLOCATE|swadmin|R|2050|4
08966|

              <sso:WorkItemTag>i2tagtest|SUMMARY|0|2</sso:WorkItemTag>
            <sso:WorkItemFields>
              <sso:vField>
                <sso:Name>TITLE</sso:Name>
                <sso:FieldType>swText</sso:FieldType>
                <sso:Value>test title 3</sso:Value>
              </sso:vField>
              <sso:vField>
                <sso:Name>TEXT</sso:Name>
                <sso:FieldType>swText</sso:FieldType>
                <sso:Value>test</sso:Value>
              </sso:vField>
              <sso:vField>
                <sso:Name>REASON</sso:Name>
                <sso:FieldType>swText</sso:FieldType>
                <sso:Value>test</sso:Value>
              </sso:vField>
              <sso:vField>
                <sso:Name>NUMERIC</sso:Name>
                <sso:FieldType>swNumeric</sso:FieldType>
                <sso:Value>123.0</sso:Value>
              </sso:vField>
              <sso:vField>
                <sso:Name>NAME</sso:Name>
                <sso:FieldType>swText</sso:FieldType>
                <sso:Value>test name 3</sso:Value>
              </sso:vField>
              <sso:vField>
                <sso:Name>MEMO</sso:Name>
                <sso:FieldType>swMemo</sso:FieldType>

```

```

        <sso:Value>test memo 4</sso:Value>
      </sso:vField>
    </sso:vField>
    <sso:Name>LOCATION</sso:Name>
    <sso:FieldType>swText</sso:FieldType>
    <sso:Value>test 6</sso:Value>
  </sso:vField>
</sso:WorkItemFields>
</sso:vWIFieldGroup>
</sso:WIFieldGroups>
</sso:vResult>
</sso:Results>
</sso:vSSOData>
</ap:SSO>
</ap:ActionResult>

```

Lock Items Request Transform XSL:

```

<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:sso="http://tibco.com/bpm/sso/types"
xmlns:ap="http://tibco.com/bpm/actionprocessor" exclude-result-
prefixes=
  "sso ap">
  <xsl:output omit-xml-declaration="yes"/>
  <xsl:param name="uniqueId" select="'_jsx_ipcNS_264_XML_ap'"/>
  <xsl:output indent="yes"/>
  <xsl:template match="/">
    <data>
      <xsl:apply-templates
select="/ap:ActionResult/ap:SSO/sso:vSSOData/
      sso:Results/sso:vResult[@Id = $uniqueId]"/>
    </data>
  </xsl:template>
  <xsl:template match="sso:vResult">
    <xsl:apply-templates select="sso:WIFieldGroups/sso:vWIFieldGroup/
      sso:WorkItemFields/sso:vField"/>
  </xsl:template>
  <xsl:template match="sso:vField">
    <xsl:element name="record">
      <xsl:attribute name="jsxid">
        <xsl:value-of select="generate-id()"/>
      </xsl:attribute>
      <xsl:attribute name="ssoName">
        <xsl:value-of select="sso:Name"/>
      </xsl:attribute>
    </xsl:element>
  </xsl:template>

```

```

        <xsl:attribute name="ssoFieldType">
            <xsl:value-of select="sso:FieldType"/>
        </xsl:attribute>
        <xsl:attribute name="ssoValue">
            <xsl:value-of select="sso:Value"/>
        </xsl:attribute>
    </xsl:element>
</xsl:template>
</xsl:stylesheet>

```

Lock Items Request CDF:

```

<data>
    <record jsxid="IDAF1CZC" ssoName="TITLE" ssoFieldType="swText"
ssoValue="test      title 3"/>
    <record jsxid="IDAJ1CZC" ssoName="TEXT" ssoFieldType="swText"
ssoValue="test"/>
    <record jsxid="IDAN1CZC" ssoName="REASON" ssoFieldType="swText"
ssoValue="test"/>
    <record jsxid="IDAR1CZC" ssoName="NUMERIC" ssoFieldType="swNumeric"
ssoValue="123.0"/>
    <record jsxid="IDAV1CZC" ssoName="NAME" ssoFieldType="swText"
ssoValue="test name 3"/>
    <record jsxid="IDAZ1CZC" ssoName="MEMO" ssoFieldType="swMemo"
ssoValue="test memo 4"/>
    <record jsxid="IDA31CZC" ssoName="LOCATION" ssoFieldType="swText"
ssoValue="test 6"/>
</data>

```

createReleaseRequest

Purpose

If a case is being started, this method creates and returns the Action Processor **StartCase** request, with the **isRelease** flag set to True.

If an existing work item is being *released* from the work queue, this method creates and returns the Action Processor **ReleaseItems** request.

This method only returns the **StartCase** or **ReleaseItems** request; it does not submit it to the Action Processor. To submit the request, you must call the **socketRequest** method (see [socketRequest](#)).

Syntax

```
createReleaseRequest(fields,  
                    requestId,  
                    validate,  
                    subProcPrecedence)
```

Parameters

Parameter	Type	Required?	Description
fields	Array of objects	Yes	- An Array of Objects containing field data properties: name, fieldType, value. - An Object, such as com.tibco.bpm.ipc.FieldData that implements <code>getFieldDataArray()</code>, which returns an Array (for information about com.tibco.bpm.ipc.FieldData, see FieldData Class).
requestId	string	No	Identifies the request. If not provided, xmlCacheIdAp is used (which can be obtained with the this.getXmlCacheIdAp function).
validate	boolean	No	If True, the field values are validated against the type to which they are defined (text, date, time, etc.). Default = False

Parameter	Type	Required?	Description
subProcPrecedence	string	No	Specifies the precedence of sub-procedure versions that are launched from the procedure. These are enumerated in <code>SWSubProcPrecedenceType</code>. Default = If not specified, the option set in the application is used (Options dialog > General tab > Sub-Case Version Precedence). Note: The <code>subProcPrecedence</code> parameter is only applicable for <code>StartCase</code>.

Returns

A string — **StartCase** or **ReleaseItems** request XML.

Remarks

This method is called by the custom form class **doRelease** method. The **Release** button on the form triggers the call to the **doRelease** method. Data is collected from the form and the XML request is obtained using this method. This request is then submitted to the Action Processor using the **socketRequest** method.

Example

The following is an example usage from **com.tibco.bpm.ipc.FormTemplate**:

```
ipcClass.prototype.doRelease = function() {
    var releaseRequest = this.createReleaseRequest(this.getFormData());
    var socket = this.socketRequest(releaseRequest);
    if (socket.isSuccess() && socket.getSsoErrorMsg() == null) {
        this.jsxsuper();
    }
};
```

Example XML

The following provides example XML for this method.

Release Request:

```
<?xml version="1.0" encoding="UTF-8"?>
<ap:Action xmlns:ap="http://tibco.com/bpm/actionprocessor"
  xmlns:sso="http://tibco.com/bpm/sso/types">
  <ap:WorkQ>
    <ap:ReleaseItems Id="_jsx_ipcNS_1394_XML_ap">
      <sso:WorkQTag>i2tagtest|swadmin|R</sso:WorkQTag>
      <sso:WIFieldData>
        <sso:vWIFieldGroup>
          <sso:WorkItemTag>i2tagtest|ALLOCATE|swadmin|R|2708|414137|
            i2tagtest|STEP1|0|2</sso:WorkItemTag>
          <sso:WorkItemFields>
            <sso:vField>
              <sso:Name>LOCATION</sso:Name>
              <sso:FieldType>swText</sso:FieldType>
              <sso:Value>USA</sso:Value>
            </sso:vField>
            ...
          </sso:WorkItemFields>
        </sso:vWIFieldGroup>
      </sso:WIFieldData>
      <sso:ValidateFields/>
    </ap:ReleaseItems>
  </ap:WorkQ>
</ap:Action>
```


Release Request (Start Case):

```
<?xml version="1.0" encoding="UTF-8"?>
<ap:Action xmlns:ap="http://tibco.com/bpm/actionprocessor"
  xmlns:sso="http://tibco.com/bpm/sso/types">
  <ap:User>
    <ap:StartCase Id="_jsx_ipcNS_217_XML_ap">
      <sso:ProcTag>i2tagtest|ALLOCATE|0|2</sso:ProcTag>
      <sso:Description>Demo</sso:Description>
      <sso:ReleaseItem>true</sso:ReleaseItem>
      <sso:ValidateFields>false</sso:ValidateFields>
      <sso:Fields>
        <sso:vField>
          <sso:Name>LOCATION</sso:Name>
          <sso:FieldType>swText</sso:FieldType>
          <sso:Value>USA</sso:Value>
        </sso:vField>
        ...
      </sso:Fields>
    </ap:StartCase>
  </ap:User>
</ap:Action>
```

doCancel

Purpose

This method closes the form dialog and unlocks the work item. No field data is saved. This method also updates the work item list.

Unlock work item and list update are not applicable for case start.

Syntax

```
doCancel()
```

Parameters

None

Returns

Void

doClose

Purpose

This method closes the form dialog and unlocks the work item. No field data is saved. This method also updates the work item list.

Unlock work item and list update are not applicable for case start.

Syntax

```
doClose()
```

Parameters

None

Returns

Void

Remarks

This method is called by the close event on the **jsx3.Dialog**.

This method calls the **doCancel** method.

doKeep

Purpose

This method closes the form dialog and updates the work item list. (List update is not applicable for case start.)

A custom form class that extends the base class must override this method. The overridden method provides the keep functionality and calls this super class method (using **this.jsxsuper();**) upon successful completion, as follows:

- The **Keep** button on the form triggers the call to the **doKeep** method.
- Data is collected from the form.
- The XML request is obtained using **createKeepRequest**. This request is then submitted to the Action Processor using **socketRequest**.
- The socket results are checked for success, and if successful, calls this super class method (using **this.jsxsuper();**).

Note that the **com.tibco.bpm.ipc.Socket** class presents the user with a message dialog if any errors are encountered. The Socket class can be checked for any error conditions using the methods shown in the example below.

Syntax

```
doKeep()
```

Parameters

None

Returns

Void

Example

The following is an example usage from **com.tibco.bpm.ipc.FormTemplate**:

```
ipcClass.prototype.doKeep = function() {
    var keepRequest = this.createKeepRequest(this.getFormData());
    var socket = this.socketRequest(keepRequest);
    if (socket.isSuccess() && socket.getSsoErrorMsg() == null) {
        this.jsxsuper();
    }
};
```

doRelease

Purpose

This method closes the form dialog and updates the work item list. (List update is not applicable for case start.)

A custom form class that extends the base class must override this method. The overridden method provides the release functionality and calls this super class method (using **this.jsxsuper();**) upon successful completion, as follows:

- The **Release** button on the form triggers the call to the **doRelease** method.
- Data is collected from the form.
- The XML request is obtained using **createReleaseRequest**. This request is then submitted to the Action Processor using **socketRequest**.
- The socket results are checked for success, and if successful, calls this super class method (using **this.jsxsuper();**).

Note that the **com.tibco.bpm.ipc.Socket** class presents the user with a message dialog if any errors are encountered. The Socket class can be checked for any error conditions using the methods shown in the example below.

Syntax

```
doRelease()
```

Parameters

None

Returns

Void

Example

The following is an example usage from **com.tibco.bpm.ipc.FormTemplate**:

```
ipcClass.prototype.doRelease = function() {  
    var releaseRequest = this.createReleaseRequest(this.getFormData());  
    var socket = this.socketRequest(releaseRequest);  
    if (socket.isSuccess() && socket.getSsoErrorMsg() == null) {  
        this.jsxsuper();  
    }  
};
```

getWindowContext

Purpose

This method provides access to the window context of the window that contains the GI Form when floating the form in a new browser window.

This method is needed because the JavaScript keyword 'window' refers back to the original iProcess browser window, not to the window context that contains the GI Form document.

Syntax

```
getWindowContext()
```

Parameters

None

Returns

The window that contains the document object to which the GI form's HTML elements are attached.

init

Purpose

This method is called when a new instance of the class is created. A custom form class that extends this base class can override this method if required, but must call the super class **init** method (using **this.jsxsuper()**). The **init** method might be overridden to set the dialog caption bar (if something other than the default value is desired) or to explicitly set the **prototypePath**.

Syntax

```
init(node,  
      strName,  
      intWidth,  
      intHeight,  
      strCaption)
```

Parameters

Parameter	Type	Required?	Description
node	jsx3.Entity	Yes	The XML node record for the selected list item.
strName	string	No	The jsx name assigned to the object. Default = "ipc.Form"
intWidth	int	No	The form dialog width. Default = 800

Parameter	Type	Required?	Description
intHeight	int	No	The form dialog height. Default = 500
strCaption	string	No	The form dialog caption. Default = see Remarks below.

Returns

Void

Remarks

The default dialog caption is as follows:

For a case start:

'Start Case: ' + description + ' - ' + procName + ' - ' + stepName

For a lock work item:

'Work Item: ' + caseNumber + ' - ' + description + ' - ' + procName + ' - ' + stepName

lockWorkItem

Purpose

This method creates an Action Processor **LockItems** request XML, submits the request, then applies the specified XSL transform to the result returned.

The **lockWorkItem** method combines several common actions into a convenient single method call. This can be used by a custom form class when initializing the form data, typically called from the custom class **postLoadInit** method override. For a code example, see **postLoadInit** (on [postLoadInit](#)).

This method returns an instance of **com.tibco.bpm.ipc.FieldData**. Each record element in the transformed CDF is added as a data field in **com.tibco.bpm.ipc.FieldData**. The field

data is read from the corresponding attributes in each record: **ssoName**, **ssoFieldType**, **ssoValue**, and **ssoIsArray**.

The **com.tibco.bpm.ipc.FieldData** class provides convenient access to field data, which can be updated and then passed into the *fields* parameter of the **createKeepRequest** or the **createReleaseRequest** function.

If an error is encountered processing the request, a null is returned.

Note that if a custom transform is applied, the returned **com.tibco.bpm.ipc.FieldData** will contain valid data only if the resulting CDF conforms to the expected default format shown below:

```
<data>
  <record ssoName="FieldName"
        ssoFieldType="swFieldType"
        ssoValue="FieldValue"
        ssoIsArray="true" or "false"/>
  ...
</data>
```

If an expected record attribute is not found, its matching property in the data object is set to null.

If record elements are not found, no data objects are created.

Syntax

```
lockWorkItem(fieldNamees,
             xslPath,
             apCacheId,
             cdfCacheId)
```

Parameters

Parameter	Type	Required?	Description
fieldNamees	array of strings or single	Yes	Specifies the fields to return from the server when locking work items. This can be specified as follows:

Parameter	Type	Required?	Description
	string		Array of strings: The names of the fields to return. Single string: Specifies the fields to return from the server using one of the following strings: “ssoFormMarkings” - Returns only visible fields/markings (based on conditional statements on the form). “ssoAllMarkings” - Returns all fields/markings, even if not visible on the form (based on conditional statements on the form). Also see, Requesting Values For Items in an Array Field.
xslPath	string	No	A file path or Cacheld for the XSL to transform the fieldDefs result into CDF. Default = Use the XSL provided in: .../JSXAPPS/ipc/components/ Forms/xsl/lockItemsToCdf.xsl
apCacheld	string	No	Cacheld for the ActionProcessor result XML. Default = Use value returned from: this.getXmlCacheldAp().
cdfCacheld	string	No	Cacheld for the transform CDF. Default = Use value returned from: this.getXmlCacheld()

Returns

Instance of **com.tibco.bpm.ipc.FieldData**.

Note: If an error is encountered processing the request, a null is returned.

onBeforeUnload

Purpose

This method can be extended by a sub-class to handle any actions to be taken before the browser window is closed directly using the window close icon. This only applies if the form is opened in a new external browser window.

Note that this method is not called if the window is closed as a result of a call to **doCancel()**.

Syntax

```
onBeforeUnload()
```

Parameters

None

Returns

Void

postLoadInit

Purpose

A custom form class that extends the base class must override this method. The **postLoadInit** method is called after the form prototype is loaded. The overridden method first calls this super class method (using **this.jsxsuper();**), then implements data loading and GUI initialization code.

Note that this method does not update (refresh) the work item list by sending a message to the server, although statuses in the list are updated (not applicable for case start).

Syntax

```
postLoadInit()
```

Parameters

None

Returns

Void

Example

The following is an example usage from **com.tibco.bpm.ipc.FormTemplate**:

```
ipcClass.prototype.postLoadInit = function() {
    this.jsxsuper();
    this.readFieldNames();
    if (! this.startCase) {
        var lockNames = new Array();
        // Returns data for the field names in the given array.
        for (var i = 0; i < this.fieldNames.length; i++) {
            if (this.fieldData.isArrayField(this.fieldNames[i])) {
                //if it is an array field, add [*] to name to return all
array values
                lockNames.push(this.fieldNames[i] + '[*]');
            }
            else {
                lockNames.push(this.fieldNames[i]);
            }
        }
        this.fieldData = this.lockWorkItem(lockNames);
        // Note: Data returned can be specified based on markings
        //       as shown below. This only applies if forms (which
        //       define the markings) exist in the procedure.
        // Return only visible fields/markings
        //       (based on conditional statements on the form).
        // this.fieldData = this.lockWorkItem("ssoFormMarkings");
        // Return all fields/markings, even if not visible on the form
        //       (based on conditional statements on the form).
        // this.fieldData = this.lockWorkItem("ssoAllMarkings");
        if (this.fieldData == null) {
            // Close the form on lock error.
        }
    }
}
```

```

        this.doCancel();
    }
    else {
        this.initializeFormData();
    }
}
else {
    this.initializeFormData();
}
}
catch (e) {
    this.showUserMessage('Error encountered loading form data: ' +
e.message);
}

```

Remarks

The following describes the sequence of calls from the **postLoadInit** method:

1. Calls the **readFieldNames** method to populate the **this.fieldNames** property, of this class, with the names of fields that are to be returned when the work item is locked.

The **readFieldNames** method calls one of the following three methods (in the **Form** base class) to obtain the names of the fields associated with a work item.

- **readFieldDefs** - This method is called if all fields defined for the procedure are desired. It submits an Action Processor request to retrieve all fields defined for the procedure.
- **readStepMarkings** - This method is called if only the fields marked on an iProcess form are desired. It submits an Action Processor request to retrieve all fields marked on an iProcess form.
- **readFormFields** - This method is called if only fields defined for a TIBCO form plug-in are desired. It submits an Action Processor request to retrieve all fields defined for a TIBCO form plug-in.

For each of the methods listed above, the field names, types, and isArray flags are returned by the Action Processor requests, and are cached in an XML CDF document, as well as stored in a **FieldData** object, which is returned to the calling function. The name of the CDF document, and the XSL transform used to generate the document, may be optionally specified as input parameters to these methods. If omitted, defaults are used.

2. Uses the **this.fieldNames** property to loop through the fields stored in the **FieldData** object (in the **this.fieldData** property of this class) to determine which fields are arrays. For each array field, “[*]” is added to the field name to signal that all array elements are to be returned when the work item is locked.
3. Calls the **lockWorkItem** method (in the **Form** base class) to lock the work item, passing the names of the fields whose values are to be returned when the item is locked.

The **lockWorkItem** method submits an Action Processor request to lock a work item and retrieve a group of fields with their values. The field names, types, and isArray flags are returned by the Action Processor requests, and are cached in an XML CDF document, as well as stored in a **FieldData** object, which is returned to the calling function. The name of the CDF document, and the XSL transform used to generate the document, may be optionally specified as input parameters to the **lockWorkItem** method. If omitted, defaults are used.

4. Calls the **initializeFormData** method to load the form with the field values returned by the **lockWorkItem** method.

The **initializeFormData** method calls the **buildCDFArrays** method to create entries, in the XML CDF document, for each element in any array fields returned by the **lockWorkItem** method. The **buildCDFArrays** method examines the cached XML CDF document, created by the **lockWorkItem** method, and identifies any array field elements (the **ssoFieldType** attribute will start with “swArrayOf” for array fields). Each array field record contains the field name and a value string containing the values of all array elements, separated by a “|” character (e.g., “a|b|c|d”). A new CDF record is created for each array element. The name of the array element includes the array index (e.g., “field[0]”), and its value is extracted from the string containing all element values.

The **initializeFormData** method then loads a GI Grid component, on the GI Form, with the resulting XML CDF document containing field names and values.

readFieldDefs

Purpose

This method creates an Action Processor **GetFieldDefs** request XML, submits the request, then applies the specified XSL transform to the result returned.

The **readFieldDefs** method combines several common actions into a convenient single method call. This is called by **com.tibco.bpm.ipc.FormTemplate.readFieldNames**. This might be most useful in development if the values of fields on a procedure are not known or if there is a need to obtain these dynamically instead of having statically defined fields on a form.

This method returns an instance of **com.tibco.bpm.ipc.FieldData**, representing all fields defined for a procedure. Each record element in the transformed CDF is added as a data field in **com.tibco.bpm.ipc.FieldData**. The field data is read from the corresponding attributes in each record: **ssoName**, **ssoFieldType**, **ssoValue**, and **ssoIsArray**.

The **com.tibco.bpm.ipc.FieldData** class provides convenient access to field data, which can be updated and then passed into the *fields* parameter of the **createKeepRequest** or the **createReleaseRequest** function.

If an error is encountered processing the request, a null is returned.

Note that if a custom transform is applied, the returned **com.tibco.bpm.ipc.FieldData** will contain valid data only if the resulting CDF conforms to the expected default format shown below:

```
<data>
  <record ssoName="FieldName"
          ssoFieldType="swFieldType"
          ssoValue="FieldValue"
          ssoIsArray="true" or "false"/>
  ...
</data>
```

If an expected record attribute is not found, its matching property in the data object is set to null.

If record elements are not found, no data objects are created.

Syntax

```
readFieldDefs(xslPath,
              apCacheId,
              cdfCacheId)
```

Parameters

Parameter	Type	Required?	Description
xslPath	string	No	A file path or Cacheld for the XSL to transform the fieldDefs result into CDF. Default = Use the XSL provided in: .../JSXAPPS/ipc/components/ Forms/xsl/fieldDefsToCdf.xsl
apCacheld	string	No	Cacheld for the ActionProcessor result XML. Default = Use value returned from: this.getXmlCacheldAp() + 'fd'
cdfCacheld	string	No	Cacheld for the transform CDF. Default = Use value returned from: this.getXmlCacheld() + 'fd'

Returns

Instance of **com.tibco.bpm.ipc.FieldData**

Note: If an error is encountered processing the request, a null is returned.

readFormFields

Purpose

This method creates an Action Processor **GetFieldDefs** request XML to retrieve the fields included in the XML returned by an Action Processor **GetPluginForm** request, submits the request, and applies the specified XSL transform to the result returned.

The **readFormFields** method combines several common actions into a convenient single method call. This is called by **com.tibco.bpm.ipc.FormTemplate.readFieldNames**. This method is useful in development if the fields defined for a TIBCO Form plug-in are not known or there is a need to obtain these dynamically instead of having statically defined fields on a form.

This method returns an instance of **com.tibco.bpm.ipc.FieldData**, representing fields defined for a TIBCO form plug-in used on a step of a procedure. Each record element in the transformed CDF is added as a data field in **com.tibco.bpm.ipc.FieldData**. The field data is read from the corresponding attributes in each record: **ssoName**, **ssoFieldType**, **ssoValue**, and **ssoIsArray**.

The **com.tibco.bpm.ipc.FieldData** class provides convenient access to field data, which can be updated and then passed into the *fields* parameter of the **createKeepRequest** or the **createReleaseRequest** function.

If an error is encountered processing the request, a null is returned.

Note that if a custom transform is applied, the returned **com.tibco.bpm.ipc.FieldData** will contain valid data only if the resulting CDF conforms to the expected default format shown below:

```
<data>
  <record ssoName="FieldName"
          ssoFieldType="swFieldType"
          ssoValue="FieldValue"
          ssoIsArray="true" or "false"/>
  ...
</data>
```

If an expected record attribute is not found, its matching property in the data object is set to null.

If record elements are not found, no data objects are created.

Syntax

```
readFormFields(xslPath,
               apCacheId,
               cdfCacheId,
               apPluginCacheId)
```


Parameters

Parameter	Type	Required?	Description
xslPath	string	No	A file path or Cacheld for the XSL to transform the fieldDefs result into CDF. Default = Use the XSL provided in: .../JSXAPPS/ipc/components/Forms/xsl/fieldDefsToCdf.xsl
apCacheld	string	No	Cacheld for the ActionProcessor result XML. Default = Use value returned from: this.getXmlCacheldAp() + 'fd'
cdfCacheld	string	No	Cacheld for the transform CDF. Default = Use value returned from: this.getXmlCacheld() + 'fd'
apPluginCacheld	string	No	Cacheld for the XML that is the result of an Action Processor GetPluginForm request. Default = Use value returned from: this.getApp().getPluginFormId()

Returns

Instance of **com.tibco.bpm.ipc.FieldData**

Note: If an error is encountered processing the request, a null is returned.

readStepMarkings

Purpose

This method creates an Action Processor **GetSteps** request XML to get fields marked on a step form, submits the request, and applies the specified XSL transform to the result returned.

The **readStepMarkings** method combines several common actions into a convenient single method call. This is called by **com.tibco.bpm.ipc.FormTemplate.readFieldNames**. This method is useful in development if the fields marked on the form of a step are not known or there is a need to obtain these dynamically instead of having statically defined fields on a form.

This method returns an instance of **com.tibco.bpm.ipc.FieldData**, representing fields marked on a form for a step in a procedure. Each record element in the transformed CDF is added as a data field in **com.tibco.bpm.ipc.FieldData**. The field data is read from the corresponding attributes in each record: **ssoName**, **ssoFieldType**, **ssoValue**, and **ssolsArray**.

The **com.tibco.bpm.ipc.FieldData** class provides convenient access to field data, which can be updated and then passed into the *fields* parameter of the **createKeepRequest** or the **createReleaseRequest** function.

If an error is encountered processing the request, a null is returned.

Note that if a custom transform is applied, the returned **com.tibco.bpm.ipc.FieldData** will contain valid data only if the resulting CDF conforms to the expected default format shown below:

```
<data>
  <record ssoName="FieldName"
        ssoFieldType="swFieldType"
        ssoValue="FieldValue"
        ssoIsArray="true" or "false"/>
  ...
</data>
```

If an expected record attribute is not found, its matching property in the data object is set to null.

If record elements are not found, no data objects are created.

Syntax

```
readStepMarkings(xslPath,  
                 apCacheId,  
                 cdfCacheId)
```

Parameters

Parameter	Type	Required?	Description
xslPath	string	No	A file path or Cacheld for the XSL to transform the fieldDefs result into CDF. Default = Use the XSL provided in: .../JSXAPPS/ipc/components/ Forms/xsl/fieldDefsToCdf.xsl
apCacheld	string	No	Cacheld for the ActionProcessor result XML. Default = Use value returned from: this.getXmlCacheldAp() + 'fd'
cdfCacheld	string	No	Cacheld for the transform CDF. Default = Use value returned from: this.getXmlCacheld() + 'fd'

Returns

Instance of **com.tibco.bpm.ipc.FieldData**

Note: If an error is encountered processing the request, a null is returned.

showUserMessage

Purpose

Displays an alert dialog with the given message. If this is a child browser window, the alert is sent to the child window context to prevent focus being moved to the parent browser window.

Note - Custom GI forms should call **this.showUserMessage** (vs. alert) to prevent window focus from shifting back to the parent window.

Syntax

```
showUserMessage(message)
```

Parameters

Parameter	Type	Required?	Description
message	string	Yes	The message to be displayed in the alert.

Returns

Void

Example

The following is an example usage:

```
this.showUserMessage(  
    'Invalid format:' +  
    '\n\n Field name: ' + fieldName +  
    '\n\n Current value: ' + fieldValue +  
    '\n\n Expected format: ' + pattern + '.');
```

socketRequest

Purpose

This method creates an Action Processor socket for the given request and submits the request to the Action Processor. This must be called to submit the request after calling any of the “**create...Request**” methods (**createKeepRequest**, **createLockRequest**, etc.).

The instance of the Socket class returned can be checked for any error conditions using the following functions:

- **socket.isSuccess** - A false value indicates an error occurred communicating with the Action Processor.
- **socket.getSsoErrorMsg** - If not null, this will contain a string message indicating what iProcess Server Object error was returned in the Action Processor result.

Syntax

```
socketRequest(requestXml,  
             cacheId)
```

Parameters

Parameter	Type	Required?	Description
requestXml	string	Yes	The Action Processor request XML.
CachedId	string	No	The CachedId where the Action Processor result is stored. Default = Use value returned from: this.getXmlCachedAp().

Returns

Instance of socket (com.tibco.bpm.ipc.Socket)

Example

The following is an example usage:

```
ipcClass.prototype.doKeep = function() {  
    var keepRequest = this.createKeepRequest(this.getFormData());  
    var socket = this.socketRequest(keepRequest);  
    if (socket.isSuccess() && socket.getSsoErrorMsg() == null) {  
        this.jsxsuper();  
    }  
};
```

Note - The **com.tibco.bpm.ipc.Socket** class presents the user with a message dialog if any errors are encountered.

transformData

Purpose

This method applies the given XSL transform to the source Cacheld XML and stores the result in the target Cacheld XML.

Syntax

```
transformData(xslPath,  
              sourceCacheId,  
              targetCacheId)
```

Parameters

Parameter	Type	Required?	Description
xslPath	string	Yes	File path or Cacheld for the transform XSL document.
sourceCacheId	string	No	Cacheld for the source XML.

Parameter	Type	Required?	Description
			Default = Use value returned from: <code>this.getXmlCacheIdAp()</code> .
<code>targetCacheId</code>	string	No	CacheId for the target XML. Default = Use value returned from: <code>this.getXmlCacheId()</code> .

Returns

A string — the result of the transform.

FieldData Class

The **`com.tibco.bpm.ipc.FieldData`** class provides convenient access to field data, which can be updated and then passed in the *fields* parameter of the **`createKeepRequest`** (see [createKeepRequest](#)) or the **`createReleaseRequest`** (see [createReleaseRequest](#)) function.

An instance of **`com.tibco.bpm.ipc.FieldData`** is returned by the following **Form** base class methods:

- [lockWorkItem](#)
- [readFieldDefs](#)
- [readStepMarkings](#)
- [readFormFields](#)

The constructor for **`com.tibco.bpm.ipc.FieldData`** does not require any parameters, for example:

```
var fieldDat = new com.tibco.bpm.ipc.FieldData();
```

For examples of usage, see **FormTemplate.js**.

Also note:

- The value of a field can be set to uninitialized by setting the `FieldData` value for the field to null.

- For all fields that are not `fieldType == 'swText'`, a zero-length string value will result in the field being set to an uninitialized state.
- Fields of `fieldType == 'swDate'` or `'swTime'` must be formatted in standard XML format. See [Date Conversions](#).

FieldData Class Functions

The **`com.tibco.bpm.ipc.FieldData`** class contains the following public functions:

- **`loadFromCdfDoc(cdfDoc)`**

This function loads `fieldDataObject` from the CDF document passed in.

where:

- `cdfDoc` is the CDF document (`jsx3.xml.Document`) from which to load the field data.

The CDF is expected to be in the following form:

```
<data>
  <record ssoName="FieldName"
          ssoFieldType="swFieldType"
          ssoValue="FieldValue"
          ssoIsArray="true" or "false"... />
  ...
</data>
```

The `fieldDataObject` serves as a map, where each association is:

- `property` = field name
- `value` = a field data Object, with properties `name`, `fieldType`, `value`, and `isArrayField`.

- **`getFieldValue(name)`**

This function returns the data value (string) for the specified field name. A null is returned if *name* is not found. If the field is an array field, this function returns an array of values, where each value in the array corresponds to an element in the array field.

where:

- *name* is the field name (string) whose value you are retrieving.

- **`getFieldType(name)`**

This function returns the field type (string) for the specified field name. A null is returned if *name* is not found.

where:

— *name* is the field name (string) whose type you are retrieving.

- **isArrayField(name)**

This function returns a Boolean indicating if the specified field is an array field. True = it is an array field.

where:

— *name* is the field name (string) in question.

- **setFieldValue(name, value)**

This function sets the value of the specified field.

where:

— *name* is the field name (string) whose value you are setting.

— *value* is the value (string) to set.

- **setFieldType(name, fieldType)**

This function sets the field type of the specified field.

where:

— *name* is the field name (string) whose field type you are setting.

— *fieldType* is the field type (string) to set.

- **addField(name, fieldType, value, isArrayField)**

This function adds a new field data Object with name, fieldType, value, and isArrayField specified. If an Object with the same name already exists, it will be replaced with the new Object.

where:

— *name* is the field name (string) to add.

— *fieldType* is the type (string) of the field to add.

— *value* is the value (string) to assign to the new field.

— *isArrayField* is a Boolean indicating if the field is an array field (True = array field).

- **removeField(name)**

This function removes the field data Object specified by setting it to null.
where:

- *name* is the field name (string) to remove.

- **getFieldDataObject()**

This function returns the fieldDataObject, which serves as a map, where each association is:

- *property* = field name
- *value* = a field data Object, with properties name, fieldType, value, and isArrayField.

- **getFieldDataArray()**

This function returns an array of Objects whose properties contain field data. These properties are: name, fieldType, value, and isArrayField.

- **enableLogging(app, forceLog)**

This function enables logging.

where:

- *app* is the application instance (com.tibco.bpm.ipc.Application) to log to.
- *forceLog* (boolean) allows you to force logging. If true, the log is written even if appLogActive = false. This is optional. Default = false.

- **log(message)**

This function logs the specified message if logging is enabled — see the **enableLogging** function.

where:

- *message* is the message to write to the log.

Requesting Values For Items in an Array Field

You can request the values for items in an array field by including an indexed string in the lock work item request array. An array field indexed string has the following syntax:

```
arrayFieldName + "[" + index + "]"
```

For example, given an array field named 'IDX_ITEM', the **lockWorkItem** method of the GI form class is called:

```
this.lockWorkItem(this.fieldNames);
```

with the `this.fieldNames` array set to include:

```
'IDX_ITEM[0]' , 'IDX_ITEM[1]' , 'IDX_ITEM[2]' ... etc
```

Array field values can also be set by including these in the `FieldData` set when calling the **doKeep** or **doRelease** methods.

Date Conversions

This section provides information about the methods available in the **com.tibco.bpm.ipc.FormDateHandler** class that allow you to convert date information between the following three formats:

- **XML** - a string used in standard serialized XML (as described at <http://www.w3.org/TR/xmlschema-2/#dateTime>)
- **JSDate** - a JavaScript Date object
- **Local** - a localized string as defined by a specified pattern

The formatting patterns use commands similar to those used for formatting in Java. The following are the date formatting tokens that are supported:

- MM - month, numeric, with leading zero
- M - month, numeric, without leading zero
- MMM - abbreviated month name
- MMMM - full month name
- dd - day number, with leading zero
- d - day number, without leading zero
- yy - year, without century, with leading zero
- yyyy - full year
- a - am/pm designation
- hh (or HH, kk, KK) - hour with leading zero
- h (or H, k, K) - hour without leading zero
- mm - minutes, always with leading zero

— ss - seconds, always with leading zero

Other rules for the formatting pattern:

- Patterns can contain tokens for date information, time information, or both.
- If the am/pm designator is included in the pattern, the hour will be interpreted as 1-12, otherwise it will be 0-23. Any of the Java-style formatting tokens for an hour can be specified, but they are all interpreted in this one manner.
- No other Java-style formatting tokens are supported (e.g., G, w, W, D, E, F, S, z, Z).
- The only alphanumeric characters that can appear in the pattern are those shown in the list above.
- Quoted, literal text is not supported in the pattern.
- One or more non-alphanumeric characters **MUST** appear between tokens as delimiters (spaces, colons, slashes, dashes, etc). There is one exception to this rule; the am/pm designator can immediately follow another token without a delimiter between.

The routines are very flexible when interpreting dates and times entered by a user, so information will not have to be entered exactly like the pattern, character by character.

- Any number of delimiting characters can be entered between parts of the date/time. These delimiting characters do not have to match the characters in the formatting pattern (keeping in mind that alpha characters cannot be used as delimiters). For instance, all of these values will be interpreted the same: '12/31/2005', '12-31-2005', '12 31 * 2005', '(12) (31?%\$#@2005)'.
- Numeric values can be entered either with or without leading zeros.
- No matter what token is used in the pattern for the month, it can be entered as either a number or as the abbreviated or full month name.
- When the name or abbreviation for a month is entered, there does not have to be delimiters around it, and it can be placed out of the expected order. However, all the remaining elements must appear in the expected order. For instance, if the pattern is 'dd-MM-yyyy', then 'Sep-10-2005', 'September 10, 2005', '10Sep2005', and even '10 2005Sep' will all be interpreted as the 10th of September, 2005.
- Time can be entered in either 24-hour or 12-hour format. If seconds or minutes are left off, they will be set to zero. The AM/PM designator does not have to appear in the exact position specified in the pattern. For instance, using any of these formatting patterns: 'h:mm:ssa', 'h:mm:ss', 'h.mm.ss a', cause all of the following

values to be interpreted as 2PM: '2:00pm', '2pm', '2:00:00pm', '14', '14:00', '14:00:00', and even 'pm2'.

- The sets of characters 'st', 'nd', 'rd', and 'th' when used with day numbers, such as '1st', '2nd', '3rd', '4th', etc., will be ignored. The fact that these follow a number will not automatically cause the number to be used as the day, however. The number must still fall in the proper relative order according to the pattern. Aside from these pairs of characters, the am/pm designator, and the full and abbreviated month names, no other alpha characters (a-z and A-Z) are allowed in the input string, and will cause a conversion to fail. Month names will, of course, have to be spelled properly, and only the one expected abbreviation will be valid.
- When 0 through 99 is entered for a year, it will be interpreted as 2000 - 2099.
- The user must enter the elements of the date/time in the order specified in the formatting pattern, but if some elements are left off the end, a suitable value will be used. For instance, when using the pattern MM-dd-yyyy, if only the month and day are entered, the current year is used. If only the month is entered, the first day of the month for the current year is used.
- Relative dates or times can be specified by entering a plus or minus sign followed by a number:
 - If the pattern includes date information, this is interpreted in days. So '+1' is tomorrow, '-1' is yesterday, '+7' is a week from today, etc. Time is set to 00:00:00.
 - For time-only fields, this is interpreted as minutes. So '-30' is thirty minutes ago, '+90' is an hour and a half from now, etc. Seconds are set to 00.
- All XML dateTime strings will have the fractional seconds values shown with a decimal followed by six zeros:
 2006-06-30T17:14:39.0000000
 Note that the JavaScript Date object milliseconds value is truncated.
- No XML dateTime strings will include a timezone value.
- The following range of dates is allowed:
 - Any valid JavaScript Date with a year of 100 or greater: 0100-01-01T00:00:00.0000000 to 275760-09-13T01:00:00.0000000
 - For a JavaScript date, this range is: 59011430400000 milliseconds (-683002 days) to 8640000000000000 milliseconds (100000000 days) elapsed since 1/1/1970 GMT.

The following are examples of valid patterns:

- MM/dd/yyyy
- dd-MMM-yyyy
- MMMM dd, yyyy
- yyyy-mm-dd
- h:mma

The following are examples of invalid patterns (although they are valid in Java patterns):

- yyyyMMddhhmmss (no delimiters)
- yyyy-mm-dd'T'hh:mm:ss.000z (quoted text and unsupported commands)
- E MMMM dd, yyyy G (unsupported commands)
- 'week:' w 'day:' F 'of' yyyy (quoted text and unsupported commands)
- h 'o'clock' a (quoted text)

Code Example

```
// Convert from JavaScript date to XML string format
var formDateHandler = new com.tibco.bpm.ipc.FormDateHandler();
var date = new Date(1182738273189);
var xmlDateString = formDateHandler.jsDateToXml(date);
alert('xmlDateString = ' + xmlDateString);
// Result: xmlDateString = 2007-06-24T19:24:33.00000000
// Convert from XML string format to display format
var pattern = 'MM/dd/yyyy h:mma'
var displayString = formDateHandler.xmlToLocal(xmlDateString,
                                                pattern);

alert('displayString = ' + displayString);
// Result: displayString = 06/24/2007 7:24pm
// Convert local string format to XML string format
var localString = '2006-12-31 23:59:59';
var pattern = 'yyyy-MM-dd hh:mm:ss';
var happyNewYear = formDateHandler.localToXml(localString,
                                                pattern);

alert('happyNewYear = ' + happyNewYear);
// Result: happyNewYear = 2006-12-31T23:59:59.00000000
```

Date Conversion Methods

The following is a list of the date conversion methods described in this section:

- [jsDateToLocal](#)
- [jsDateToXml](#)
- [localToJSDate](#)
- [localToXml](#)
- [xmlToJSDate](#)
- [xmlToLocal](#)

jsDateToLocal

Purpose

This method converts a JavaScript Date object into a string containing a formatted local date value.

Syntax

```
jsDateToLocal(jsDate, pattern)
```

Parameters

Parameter	Type	Required?	Description
jsDate	Date object	Yes	JavaScript Date object to be converted.
pattern	string	Yes	A valid format pattern — specifies the format of the date string returned by this method.

Returns

A string containing a formatted date. Returns null if parameters are omitted or the format pattern is invalid.

Remarks

If the date portion of the format pattern is omitted, the resulting date is 12/30/1899. If the time portion of the format pattern is omitted, the resulting time is 00:00.

jsDateToXml

Purpose

This method converts a JavaScript Date object into a standard serialized XML date string.

Syntax

```
jsDateToXml(jsDate)
```

Parameters

Parameter	Type	Required?	Description
jsDate	Date object	Yes	JavaScript Date object to be converted.

Returns

A string containing a date in standard serialized XML format:

```
yyyy-MM-ddThh:mm:ss.0000000
```

Returns null if the jsDate parameter is omitted.

localToJSDate

Purpose

This method converts a string containing a formatted local date value into a JavaScript Date object.

Syntax

```
localToJSDate(strDate, pattern)
```

Parameters

Parameter	Type	Required?	Description
strDate	string	Yes	Formatted date string.
pattern	string	Yes	Valid format pattern — the format of strDate parameter.

Returns

A JavaScript Date object. Returns null if parameters are omitted or the format pattern is invalid.

Remarks

If the date portion of the format pattern is omitted, the resulting date is 12/30/1899. If the time portion of the format pattern is omitted, the resulting time is 00:00.

localToXml

Purpose

This method converts a string containing a formatted local date value into a standard serialized XML date string.

Syntax

```
localToXml(strDate, pattern)
```

Parameters

Parameter	Type	Required?	Description
strDate	string	Yes	Formatted date string.
pattern	string	Yes	Valid format pattern — the format of strDate parameter.

Returns

A string containing a date in standard serialized XML format:

yyyy-MM-ddThh:mm:ss.0000000

Returns null if parameters are omitted or the format pattern is invalid.

Remarks

If the date portion of the format pattern is omitted, the resulting date is 12/30/1899. If the time portion of the format pattern is omitted, the resulting time is 00:00.

xmlToJSDate

Purpose

This method converts a standard serialized XML date string into a JavaScript Date object.

Syntax

```
xmlToJSDate(xmlDate)
```

Parameters

Parameter	Type	Required?	Description
xmlDate	string	Yes	Standard serialized XML date string in the format: yyyy-MM-ddThh:mm:ss.0000000

Returns

A JavaScript Date object. Returns null if the xmlDate parameter is omitted or the format pattern is invalid.

xmlToLocal

Purpose

This method converts a standard serialized XML date string into a string containing a formatted local date value.

Syntax

```
xmlToLocal(xmlDate, pattern)
```

Parameters

Parameter	Type	Required?	Description
xmlDate	string	Yes	Standard serialized XML date string in the format: yyyy-MM-ddThh:mm:ss.0000000
pattern	string	Yes	Valid format pattern - specifies the format of the date string returned by this method.

Returns

A string containing a formatted date. Returns null if parameters are omitted or the format pattern is invalid.

Remarks

If the date portion of the format pattern is omitted, the resulting date is 12/30/1899. If the time portion of the format pattern is omitted, the resulting time is 00:00.

Date Format Localization Methods

This section provides information about the methods available in the **com.tibco.bpm.ipc.FormDateHandler** class that allow you to customize text strings representing months and the am/pm designation. These are used when matching a format pattern converting to or from a local date string format.

- [getAbbrMonthNames](#)
- [setAbbrMonthNames](#)
- [getAmPm](#)
- [setAmPm](#)
- [getFullMonthNames](#)
- [setFullMonthNames](#)

getAbbrMonthNames

Purpose

This method returns the abbreviations of the months that are returned when the abbreviated month name format (MMM) is specified in a pattern parameter to a date conversion method.

Syntax

```
getAbbrMonthNames()
```

Parameters

None

Returns

JavaScript array of strings representing the abbreviations for months of the year, in order from the first month to the twelfth month.

Remarks

The default abbreviated month names returned by the date conversion methods are: 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'.

setAbbrMonthNames

Purpose

This method allows you to specify the abbreviations that are returned when the abbreviated month name format (MMM) is specified in a pattern parameter to a date conversion method.

Syntax

```
setAbbrMonthNames(monthNames)
```

Parameters

Parameter	Type	Required?	Description
monthNames	JavaScript array	Yes	Array of strings representing the abbreviations for the months of the year, in order from the first month to the twelfth month.

Returns

None

Remarks

The default abbreviated month names returned by the date conversion methods are: 'Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'.

getAmPm

Purpose

This method returns the am/pm designation that is returned when the am/pm designator ('a') is specified in a pattern parameter to a date conversion method.

Syntax

```
getAmPm()
```

Parameters

None

Returns

JavaScript array of two strings, the first representing the am, and the second representing the pm designation.

Remarks

The default designations returned by the date conversion methods are: 'am', 'pm'.

setAmPm

Purpose

This method allows you to specify the am/pm designation that is returned when the am/pm designator ('a') is specified in a pattern parameter to a date conversion method.

Syntax

```
setAmPm(designators)
```

Parameters

Parameter	Type	Required?	Description
designators	JavaScript array	Yes	Array of strings representing the am/pm designators, the first element represents am, and the second represents pm.

Returns

None

Remarks

The default designations returned by the date conversion methods are: 'am', 'pm'.

getFullMonthNames

Purpose

This method returns the names of the months that are returned when the full month name format (MMMM) is specified in a pattern parameter to a date conversion method.

Syntax

```
getFullMonthNames()
```

Parameters

None

Returns

JavaScript array of strings representing the months of the year, in order from the first month to the twelfth month.

Remarks

The default full month names returned by the date conversion methods are: 'January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December'.

setFullMonthNames

Purpose

This method allows you to specify the names of the months that are returned when the full month name format (MMMM) is specified in a pattern parameter to a date conversion method.

Syntax

```
setFullMonthNames(monthNames)
```

Parameters

Parameter	Type	Required?	Description
monthNames	JavaScript array	Yes	Array of strings representing the months of the year, in order from the first month to the twelfth month.

Returns

None.

Remarks

The default full month names returned by the date conversion methods are: 'January', 'February', 'March', 'April', 'May', 'June', 'July', 'August', 'September', 'October', 'November', 'December'.

Accessing User Options When Using GI Forms

User options establish default settings for each user who logs into the iProcess Workspace (Browser). These include things such as the language to display, form position on the

screen, etc. (More information about the options available and how they can be set from the application can be found in the *iProcess Workspace (Browser) User's Guide*.)

The custom GI form class that extends the **com.tibco.bpm.ipc.Form** base class can access the user options for the currently logged in user using:

```
this.getAppPrefValue(prefName)
```

where *prefName* identifies the user option for which you would like the value. This method returns a string, identifying the current setting of the given user option. The following table lists the valid *prefName* values and the possible return values for each option (note that *options* were formerly called *preferences*, hence the *prefName* parameter):

prefName	Return Values	Description
language	"language name"	Identifies the language in which the iProcess Workspace (Browser) is displayed. The default is "English(US)".
formLeft	"integer value"	The work item form window is positioned this number of pixels from the left. (Only applicable if both formFullscreen and formCenter are false.)
formTop	"integer value"	The work item form window is positioned this number of pixels from the top. (Only applicable if both formFullscreen and formCenter are false.)
formWidth	"integer value"	The width (in pixels) of the work item form window. (Only applicable if formFullscreen is false.)
formHeight	"integer value"	The height (in pixels) of the work item form window. (Only applicable if formFullscreen is false.)
formFullscreen	"true"	The work item form window is displayed full screen.

prefName	Return Values	Description
	“false”	The work item form window is not displayed full screen. (Other options are used to determine position/size.)
formCenter	“true”	The work item form window is displayed centered. (the formWidth and formHeight options are used to determine size.)
	“false”	The work item form window is not displayed centered. (Other options are used to determine position/size.)
subProcPrecedence	“swPrecedenceR”	The precedence in which sub-procedures are started from the main procedure:
	“swPrecedenceUR”	— swPrecedenceR: Only released sub-procedures are started.
	“swPrecedenceMR”	— swPrecedenceUR: Unreleased, then released.
	“swPrecedenceUMR”	— swPrecedenceMR: Model, then released.
	“swPrecedenceMUR”	— swPrecedenceUMR: Unreleased, model, then released.
		— swPrecedenceMUR: Model, unreleased, then released.

ASP Forms

This section describes how to set up the ASP form example that is provided with the iProcess Workspace (Browser).

ASP Form Example

A Microsoft Visual Studio .NET 2003 project that defines an example ASP form is provided with the iProcess Workspace (Browser). This example can be used as a starting point to create your own custom ASP form project.

The example .NET project is located in the following directory:

```
InstallationHomeDir\iprocessclientbrowser\samples\ASPFormExample
```

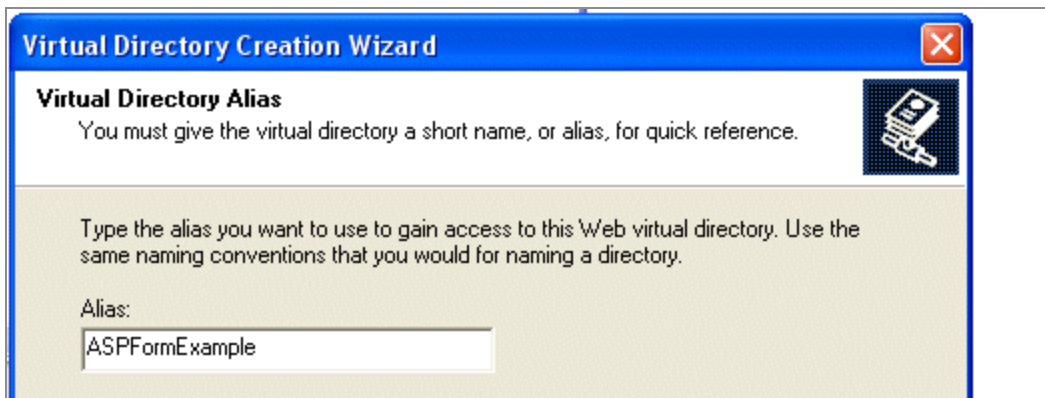
where *InstallationHomeDir* is the directory in which the installer places administrative files, such as the uninstaller, documentation, and sample code. This defaults to C:\tibco on Windows systems, and /opt/tibco on UNIX systems, but can be specified as a different directory when the iProcess Workspace (Browser) is installed.

The following sections describe how to implement this example.

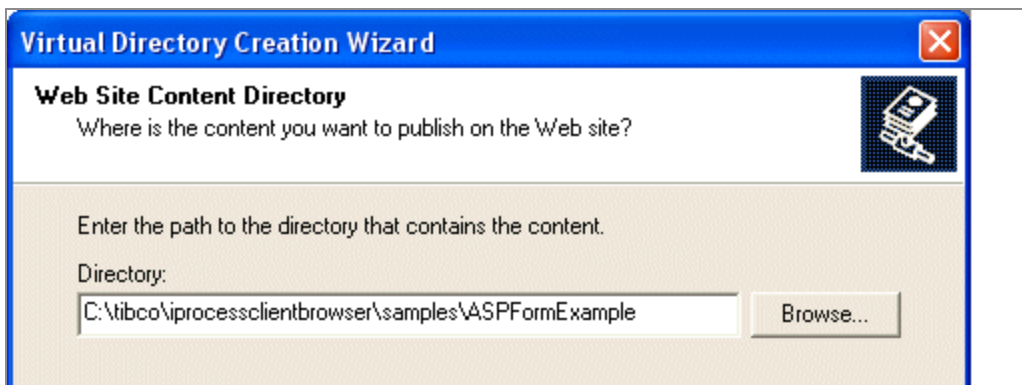
Setting Up the ASP Form Project in IIS

Perform the following steps to set up the Microsoft Visual Studio .NET 2003 ASP form project in Microsoft Internet Information Services (IIS):

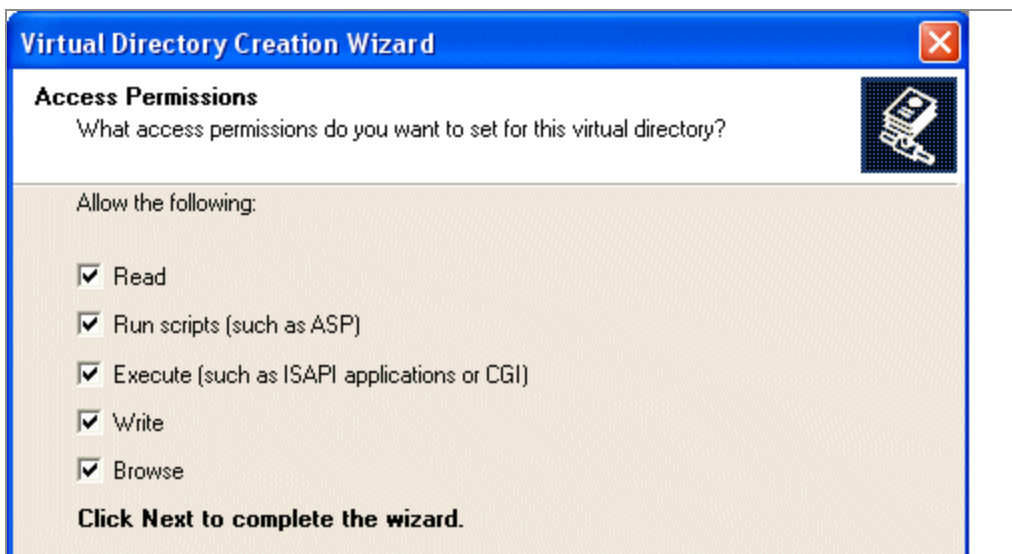
1. Create a virtual directory in IIS and give it an alias name. In this example, we will use the name of the example project.



2. Set the physical directory to point to the location of the ASPFormExample .NET project.



3. Select all permissions for development purposes. The permissions on the finished production forms can be set to secure settings at a later time.



4. Start IIS if it is not already running.

5. Open Visual Studio .NET 2003 and select **File > Open > Project From Web**, then enter the URL to the project. The path will be:

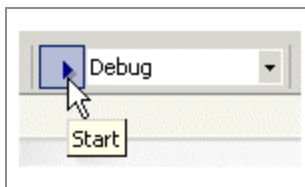
```
http://Host:Port/ASPFormExample
```

where:

- *Host* is the name of the machine on which you've installed the ASP form project.
- *Port* is the port used by IIS to communicate with web applications.

The **Open Project** dialog is displayed.

6. Select and open the `ASPFormExample.csproj` file.
7. In Visual Studio, select **Build > Rebuild Solution**.
You will be prompted to save the solution file (`.sln`).
8. Save the solution file in the `ASPFormExample` directory.
9. In Visual Studio, right click on `ASPForm.aspx` in the **Solution Explorer** window and select **Set As Start Page**.
10. In Visual Studio, select **Debug** in the **Solution Configuration** drop-down list, then click the start arrow to the left of the field.

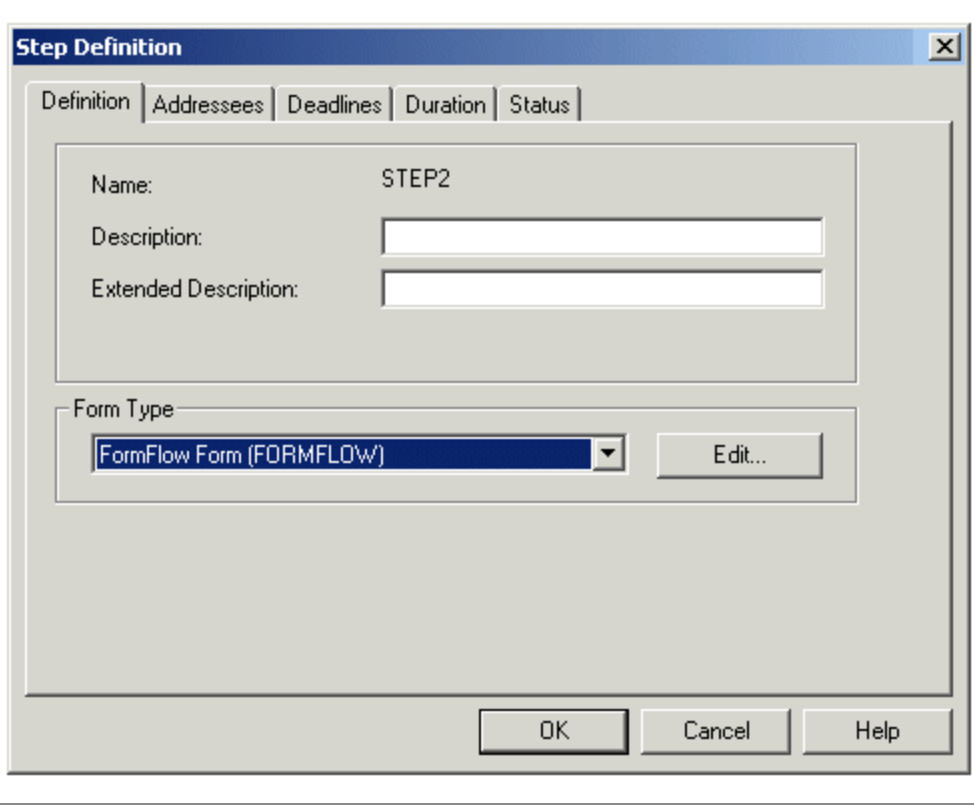


Visual Studio will connect to IIS, allowing you to develop and debug the **ASPFormExample** project. (A browser window may appear with the `ASPForm.aspx` page displayed containing an error message — you can ignore this message and close the browser window.)

Configuring iProcess Workspace (Browser) to Use the ASP Form

Perform the following steps to configure the iProcess Workspace (Browser) to use the ASP form:

1. Set the **ExternalFormURI** parameter in the Action Processor's configuration file, `apConfig.xml`. This specifies the base URL of the Web Application Server (IIS in this case) that is hosting your ASP Forms. For information about this parameter, see [External Form URI](#).
2. Create a procedure and define a normal step, or import `ASPForm.xfr` (which is included in the **ASPFormExample** project).
3. Set the form type of the normal step to "FormFlow Form".

The image shows a "Step Definition" dialog box with a blue title bar and a close button. It has five tabs: "Definition", "Addressees", "Deadlines", "Duration", and "Status". The "Definition" tab is active. Inside the dialog, there are three text input fields: "Name:" with the value "STEP2", "Description:", and "Extended Description:". Below these is a "Form Type" section with a dropdown menu showing "FormFlow Form (FORMFLOW)" and an "Edit..." button. At the bottom are "OK", "Cancel", and "Help" buttons.

4. Click the **Edit** button on the **Step Definition** dialog. The **FormFlow Form** dialog is displayed.
5. Enter the location of the `ASPForm.aspx` file. Don't enter the full URL, as the base URL location is defined in the **ExternalFormURI** parameter (see step 1). Only enter the portion of the URL that is unique to the step.

STEP2 (): FormFlow Form

FormFlow Form

FormFlow Reference

FormFlow Form:

Field References

☒ Permit access to all fields from FormFlow form.

☐ Permit access to only specific fields from FormFlow form...

Field	Description	Access	Type	Length
▶	▼			

OK Cancel

6. Start a case of the procedure. The ASP form example should look as follows.

The screenshot shows a web browser window titled "TIBCO Web Client ASP.NET Form Example - Microsoft Internet Explorer". The page content is titled "TIBCO Web Client ASP Form Example". Below the title, there are seven input fields, each with a label to its left: TEXTFLD1, TEXTFLD2, TEXTFLD3, NUMERIC1, CNUMERIC1, DATE1, and TIME1. Each label is followed by a text input box. At the bottom left of the form, there is a blue button labeled "Start Case".

7. Edit the **ASPFormExample** example project for the desired form layout and fields to be displayed.

Define the field names in the **fieldNames** array, the types in the **fieldTypes** array, and the date format in the **dateFormat** string.

The field names should correspond to the iProcess Engine procedure field names. The arrays are defined in ASPForm.aspx, as follows:

```
String [] fieldNames = {"TEXTFLD1", "TEXTFLD2", "TEXTFLD3",
                        "NUMERIC1", "CNUMERIC1", "DATE1", "TIME1"};
String [] fieldTypes = {"swText", "swText", "swText", "swNumeric",
                        "swComma", "swDate", "swTime"};
String dateFormat = "MDY";
```

The **dateFormat** variable can be set to "MDY", "DMY", or "YMD", to indicate the order of the day, month, and year in date fields. (Time fields will be displayed in the hh:mm format.)

The position and look of these fields can be defined in the `ASPForm.css` cascading style sheet. For example:

```
#TEXTFLD1marking {
    position:relative;
    left:75px;
    top:0px;
    width:240px;
}
```

ASP Form Interface

The `ASPForm.aspx` form makes use of the interfaces defined in the `ASPFormLib.cs` library in order to do a start case, lock item, release item, keep item, and undo item. These interfaces construct and make the request to the Action Processor. The **ASPFormLib** public interfaces available are as follows:

ASPFormLib Constructor

The constructor initiates the **ASPFormLib** with the request and field information.

```
public ASPFormLib (HttpRequest aRequest,
                  string [] aFieldNames,
                  string [] aFieldTypes)
```

getRequestType

This method returns the type of the request.

```
public int getRequestType()
```

The **getRequestType** method returns one of the following constant int values:

- `public const int REQUEST_TYPE_UNKNOWN = 0x0;`
- `public const int REQUEST_TYPE_RENDER_WORK_ITEM_FORM = 0x1;`
- `public const int REQUEST_TYPE_RENDER_START_CASE_FORM = 0x2;`
- `public const int REQUEST_TYPE_START_CASE = 0x3;`

- `public const int REQUEST_TYPE_UNDO = 0x4;`
- `public const int REQUEST_TYPE_KEEP = 0x5;`
- `public const int REQUEST_TYPE_RELEASE = 0x6;`

getInitXML

This method returns the details of the XML generated by an **Undo**, **Keep** or **Release** button click on the form.

```
public string getInitXML()
```

startCase

This method creates and submits an Action Processor **StartCase** request.

```
public void startCase()
```

undoItem

This method creates and submits an Action Processor **UndoItems** request.

```
public void undoItem()
```

lockItem

This method creates and submits an Action Processor **LockItems** request.

```
public String [] lockItem()
```

keepItem

This method creates and submits an Action Processor **KeepItems** request.

```
public void keepItem()
```

releaseItem

This method creates and submits an Action Processor **ReleaseItems** request.

```
public void releaseItem()
```

JSP Forms

This section describes how to set up the JSP form example that is provided with the iProcess Workspace (Browser).

JSP Form Example

A JSP form example is provided with the iProcess Workspace (Browser). This example can be used as a starting point to create your own custom JSP form project.

The example is in the form of an IntelliJ project that was developed using IntelliJ IDEA 7.0.4 and assumes Tomcat as the application server.

You can open and build the project using IntelliJ, or you can extract the files from the `\src` directory and create a project using those files in whatever Java development tool you desire.

The example IntelliJ project is located in the following directory:

```
InstallationHome\iprocessclientbrowser\Samples\JSPFormExample
```

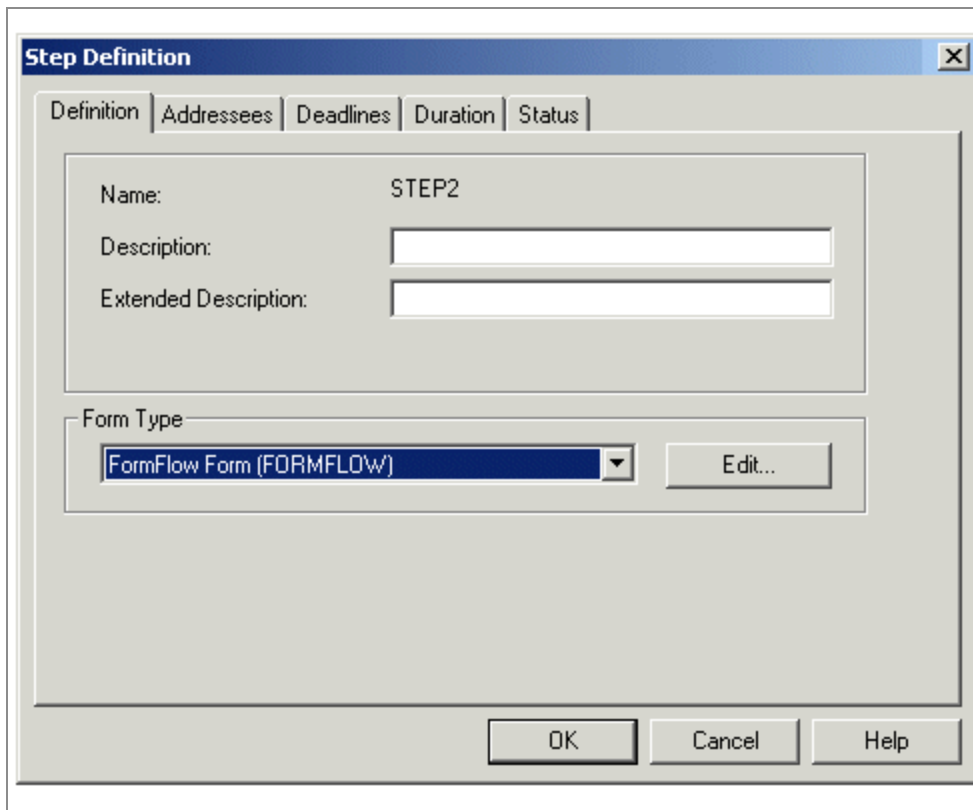
where *InstallationHome* is the directory in which the installer places administrative files, such as the uninstaller, documentation, and sample code. This defaults to `C:\tibco` on Windows systems, and `/opt/tibco` on UNIX systems, but can be specified as a different directory when the iProcess Workspace (Browser) is installed.

If you build the project in IntelliJ, move the resulting `JSPFormExample.war` file to the `webapps` directory on Tomcat and start Tomcat. The form example will extract itself into a directory called `JSPFormExample` under *TomcatHome*\webapps.

If you build the project in a different Java development tool, or use an application server other than Tomcat, refer to the instructions for those tools for building and extracting `.war` files.

Configure iProcess Workspace (Browser) to Use the JSP Form

1. Set the **ExternalFormURI** parameter in the Action Processor's configuration file, **apConfig.xml**. This specifies the base URL of the Web Application Server (Tomcat in this case) that is hosting your JSP Forms. For information about this parameter, see [External Form URI](#).
2. Create a procedure and define a normal step, or import JSPForm.xfr (which is included in the **JSPFormExample** project).
3. Set the form type of the normal step to "FormFlow Form".



The screenshot shows the "Step Definition" dialog box. It has a title bar with a close button. Below the title bar are five tabs: "Definition", "Addressees", "Deadlines", "Duration", and "Status". The "Definition" tab is selected. Inside the dialog, there are three text input fields: "Name:" with the value "STEP2", "Description:", and "Extended Description:". Below these fields is a "Form Type" section with a dropdown menu showing "FormFlow Form (FORMFLOW)" and an "Edit..." button. At the bottom of the dialog are three buttons: "OK", "Cancel", and "Help".

4. Click the **Edit** button on the **Step Definition** dialog. The **FormFlow Form** dialog is displayed.
5. Enter the location of the JSPForm.jsp file. Don't enter the full URL as the base URL location is defined in the **ExternalFormURI** parameter (see step 1). Only specify the portion of the URL that is unique to the step.

STEP2 (): FormFlow Form

FormFlow Form

FormFlow Reference

FormFlow Form: JSPFormExample/JSPForm.jsp

Field References

☒ Permit access to all fields from FormFlow form.

☐ Permit access to only specific fields from FormFlow form...

Field	Description	Access	Type	Length

OK Cancel

6. Start a case of the procedure. The JSP form example should look as follows.

TIBCO Web Client JSP Form Example

TEXTFLD1 :

TEXTFLD2 :

TEXTFLD3 :

NUMERIC1 :

CNUMERIC1 :

DATE1 :

TIME1 :

Start Case

7. Edit the **JSPFormExample** example project for the desired form layout and fields to be displayed.

Define the field names in the **fieldNames** array, the types in the **fieldTypes** array, and the date format in the **dateFormat** string.

The field names should correspond to the iProcess Engine procedure field names. The arrays are defined in JSPForm.jsp, as follows:

```
String [] fieldNames = {"TEXTFLD1", "TEXTFLD2", "TEXTFLD3",
                        "NUMERIC1", "CNUMERIC1", "DATE1", "TIME1"};
String [] fieldTypes = {"swText", "swText", "swText", "swNumeric",
                        "swComma", "swDate", "swTime"};
String dateFormat = "MDY";
```

The **dateFormat** variable can be set to “MDY”, “DMY”, or “YMD”, to indicate the order of the day, month, and year in date fields. (Time fields will be displayed in the hh:mm format.)

The position and look of these fields can be defined in the `JSPForm.css` cascading style sheet. For example:

```
#TEXTFLD1marking {
    position:relative;
    left:75px;
    top:0px;
    width:240px;
}
```

JSP Form Interface

The `JSPForm.jsp` form makes use of the interfaces defined in the `JSPFormLib.java` library in order to do a start case, lock item, release item, keep item, and undo item. These interfaces construct and make the request to the Action Processor. The **JSPFormLib** public interfaces available are as follows:

JSPFormLib Constructor

The constructor initiates the **JSPFormLib** with the request and field information.

```
public JSPFormLib (HttpServletRequest aRequest,
                  String [] aFieldNames,
                  String [] aFieldTypes)
```

getRequestType

This method returns the type of the request.

```
public int getRequestType()
```

The **getRequestType** method returns one of the following static int values:

- `public static final int REQUEST_TYPE_UNKNOWN = 0x0;`
- `public static final int REQUEST_TYPE_RENDER_WORK_ITEM_FORM = 0x1;`
- `public static final int REQUEST_TYPE_RENDER_START_CASE_FORM = 0x2;`
- `public static final int REQUEST_TYPE_START_CASE = 0x3;`

- `public static final int REQUEST_TYPE_UNDO = 0x4;`
- `public static final int REQUEST_TYPE_KEEP = 0x5;`
- `public static final int REQUEST_TYPE_RELEASE = 0x6;`

getInitXML

This method returns the details of the XML generated by an **Undo**, **Keep** or **Release** button click on the form.

```
public String getInitXML()
```

startCase

This method creates and submits an Action Processor **StartCase** request.

```
public void startCase()
```

undoItem

This method creates and submits an Action Processor **UndoItems** request.

```
public void undoItem()
```

lockItem

This method creates and submits an Action Processor **LockItems** request.

```
public String [] lockItem()
```

keepItem

This method creates and submits an Action Processor **KeepItems** request.

```
public void keepItem()
```

releaseItem

This method creates and submits an Action Processor **ReleaseItems** request.

```
public void releaseItem()
```

Customizing iProcess Modeler Forms

This section describes how to customize TIBCO iProcess Modeler-generated forms using additional HTML and scripting code.

Overview

The TIBCO iProcess Workspace (Browser) renders the standard iProcess Modeler forms as HTML with scripting. This can be customized with additional HTML and scripting code to alter the appearance and augment the functionality provided by the standard form.

Customizations can be done using two methods:

- **Embedding** - Additional HTML can be embedded directly into the iProcess Modeler Form definition so that it is included when the browser form is dynamically generated.
 - The customizations are stored in the form definitions. Exporting and importing the procedure maintains the customizations.
 - A smaller, simpler set of scripting functions are used to access work item data.
 - The iProcess Modeler Form definition will not be suitable for display in the iProcess Workspace (Windows), where the HTML tags and scripting code will be visible as part of the form.
 - Each type of marking (required marking, optional marking, display field, and embedded field) has a consistent, standard appearance. You can customize the appearance of each type of control, but not individual markings. You can, however, create your HTML controls, then set and get work item / case values.
 - Standard **Undo**, **Keep**, and **Release** buttons always appear at the bottom of the form.
- **File Caching** - The HTML that is generated for the iProcess Modeler Form can be altered and saved to a disk cache where it will be used instead of dynamically generated HTML.

- The customizations are stored in files on the web server and will not automatically follow the iProcess Modeler Form definition through export / import, so backups will need to be maintained separately.
- Customizations can be saved for use with a specific minor version of the procedure, a specific major version, or for all versions. So when the procedure is changed, new cache files may need to be created, copied, or modified so that the appropriate customizations are available for use with a modified procedure definition, or for work items not migrated to the new version of the procedure.
- The iProcess Modeler Form definition can be designed to work well in iProcess Workspace (Windows), while the customizations are used only for the iProcess Workspace (Browser).
- The appearance and functionality of the individual marking control can be customized.
- The **Undo**, **Keep**, and **Release** buttons can be modified or moved. They can also be removed and replaced with other controls that would trigger the actions through scripting.

If you will only be displaying forms in the iProcess Workspace (Browser), and your customization needs are not overly complex, you might choose to embed the HTML into the form definition. If you need to be able to use both the rich client and browser versions of the iProcess Workspace (Browser), or apply more extensive customizations, you might choose file-cached customizations.

Also, you do not have to handle all procedures, steps, and versions of steps using the same method. You can use a mixture of standard dynamically generated HTML, embedded HTML, and file-cached HTML.

These methods of customization are described in more detail in the following sections.

Embedding HTML

When embedding HTML directly into the iProcess Modeler Form definition, any valid HTML can be included, with a few limitations and considerations, as described in the following subsections.

Word Wrap in the Editor

The editor for the iProcess Modeler Form limits the line length for entering text into the form definition. When typing or pasting HTML code into the form, you may want to set the line length to the maximum value of 128. If the editor does wrap code to a new line, you may need to alter it so that line breaks do not adversely affect the HTML.

Pre-Formatting of the Form

By default, all the text that appears in the form definition is treated as one large block of text with spacing preserved and carriage returns between the lines. Where fields appear in the iProcess Modeler Form, one or more HTML elements will be inserted into that block of text. By default, this block of HTML is enclosed inside an HTML **<pre>** tag. This preserves all whitespace in the form, including multiple spaces and line feeds. It also limits fonts to monospaced fonts.

Disabling Pre-Formatting

Pre-formatting could interfere with the appearance of some types of HTML you embed. If it does, the pre-formatting can be disabled by including the HTML comment “<!-- DISABLE PRE -->” anywhere in the form definition. When that comment is detected, the HTML **<pre>** tag will not be placed around the HTML for the form. If there are some isolated sections of your form that still need to be pre-formatted, **<pre>** tags can be added where needed in the form definition.

Including Scripts

Scripting can also be included in the form. As an example, to disable the context menu for the page, you could include the following block of text anywhere on the form.

```
<script language="javascript">  
    document.oncontextmenu = function (){return false;}  
</script>
```

Nesting of HTML Tags with Conditional Statements

Where the iProcess Modeler Form definition includes IF / ELSE / ENDIF conditions, the blocks affected by a condition are enclosed in an HTML **** tag. Make sure that any opening and closing HTML tags that you add nest properly with these. For instance, you should not put a **** tag on the line before an IF statement and the matching **** tag on the line right after the IF. You could, however, put the matching **** tag after the ENDIF.

Functions Available for Embedded Scripting

Two **.js** files are imported into the page generated for an iProcess Modeler Form:

- **spddate.js** - Contains functions for date, time, and numeric data conversions.
- **spdform.js** - Contains functions for accessing the work item / case data and performing other interactions with the iProcess Modeler Forms.

Comments inside of **spddate.js** describe the available functions. The file also contains many private variables and functions. The names of these private variables and functions all begin with the text “_private”. You should not call or access these private items as they are subject to change or removal in future versions.

Comments inside of **spdform.js** describe the functions that are available for use from scripting. The ones that are specifically for use with HTML embedded directly into the form definition are labeled with the text “Supported in Client Scripting”. Do not use other functions in the file as they are only for use with file-cache customizations, or are private functions that are subject to change or removal in future versions.

These are the functions allowed for use in embedded scripting:

- **spdSetFieldValue**(fieldName, fvalue, spdFormat, spdPreDecimalDigits)
- **spdGetFieldValue**(fieldName, spdFormat, getInitialValue)
- **spdGetWorkItemTag**()
- **spdGetProcTag**()
- **spdIsFieldValid**(fieldName)
- **spdGetFieldErrorMsg**(fieldName)
- **spdSetFieldNotificationFunction**(fieldName, notificationFunction)

Use the **spdSetFieldValue()** and **spdGetFieldValue()** functions to get and set the values of any work item or case fields whether or not the field is included as a marking. Any change to the field value through these functions will update the information displayed in marking controls on the form.

Separate functions (**spdGetWorkItemTag()** and **spdGetProcTag()**) exist for retrieving the work item tag and procedure tag, since these are not actually accessible as case fields.

The **spdIsFieldValid()** function returns false if invalid data is entered on the form for a field, for example, a date with a value of “13/13/2006”. When the field data is not valid, the field error message will contain a text description of the problem.

The **spdSetFieldNotificationFunction()** function lets you specify a function to be called when the value of the specified field changes. The function will also be called during initialization of the form if the form includes a marking for the field (including hidden or embedded markings).

Altering the Style of Various Controls

The appearance of the various marking controls is set through a cascading style sheet named **spdform.css**. You should not directly alter this file, but can include styles to override the values directly in your form definition or add a reference to an additional CSS file to override settings.

The following are the class names for the styles used in the dynamically generated iProcess Modeler Forms:



Note

You’ll see the “SPD” acronym used in places. This is a carryover from the previously used “Staffware Process Definer” name, which is now called the “iProcess Modeler”.

- **SPD_CONTAINER** - Style for an area containing the entire iProcess Modeler Form. Used to create a border, padding default color and font information.
- **SPD_MARKING_REQ** - Style applied to input controls (text input or select) for required markings.
- **SPD_MARKING_OPT** - Style applied to input controls (text input or select) for optional markings.
- **SPD_MARKING_EMBEDDED** - Style applied to embedded marking data.

- **SPD_MARKING_READONLY** - Style applied to read-only marking data.
- **SPD_MEMO_BUTTON_REQ** - Style applied to memo buttons for editing required memo fields.
- **SPD_MEMO_BUTTON_OPT** - Style applied to memo buttons for editing optional memo fields.
- **SPD_MEMO_BUTTON_READONLY** - Style applied to memo buttons for displaying read-only memo fields.
- **SPD_MESSAGE** - Style applied to a message area that could appear at the top of the form (rarely shown).
- **SPD_BUTTON** - Style applied to the buttons used to trigger the undo, keep, and release actions.
- **SPD_HELP** - Style applied to area/link for displaying field help.
- **SPD_CALENDAR** - Style applied to the area/link for displaying the popup calendar.
- **SPDCAL_CONTAINER** - Style for an area containing a popup calendar (used to set border/padding/ default color/font).
- **SPDCAL_PREV** - Style for the area/link for going to the previous year and month.
- **SPDCAL_NEXT** - Style for the area/link for going to the next year and month.
- **SPDCAL_HEADER** - Style for the heading areas displaying the year and month name.
- **SPDCAL_WEEKDAYS** - Style for the area displaying the abbreviated day names.
- **SPDCAL_ACTIVEDAY** - Style for all selectable days from the displayed month.
- **SPDCAL_INACTIVEDAY** - Style for all the non-selectable days before/after the display month.
- **SPDCAL_SELECTEDDAY** - Style for the currently selected day (or current date if none was already selected).

For example, to alter the background color for optional fields (only used if the field is empty), you could include the following in the iProcess Modeler Form definition:

```
<style>
<!--
.SPDCAL_MARKING_OPT {background-color: #008800 }
-->
</style>
```

Embedded Customization Examples

The file **IPCBrowserExamples.xpd** contains several example procedures illustrating embedded HTML. This file is located in the following directory:

InstallationHomeDir\iprocessclientbrowser\samples\IPCBrowserFormExamples

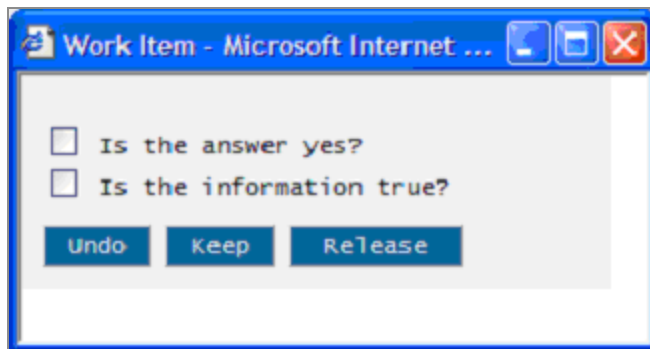
where *InstallationHomeDir* is the directory in which the installer places administrative files, such as the uninstaller, documentation, and sample code. This defaults to *C:\tibco* on Windows systems, and */opt/tibco* on UNIX systems, but can be specified as a different directory when the iProcess Workspace (Browser) is installed.

The examples provided are described in the following subsections.

CHECKBOX

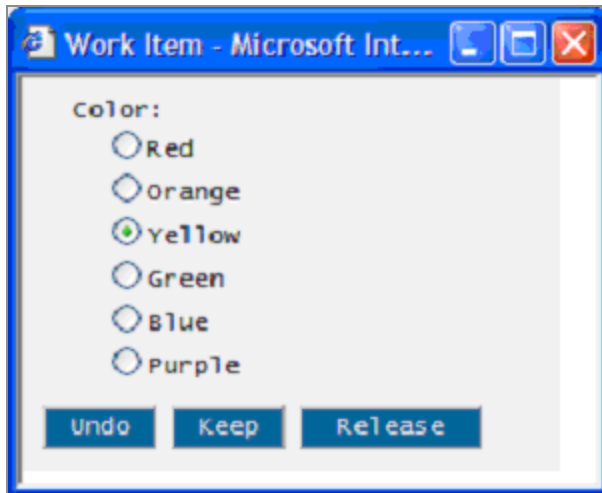
The CHECKBOX example uses check boxes to modify field data.

- The form includes a hidden marking for each of the fields that will be edited with a check box.
- An input control of the type “checkbox” is included for each field. The **onclick** event of the check box calls the function **spdSetFieldValue()** with the appropriate value for the current checked state of the control.
- A notification function is defined and is registered with the `spdform.js` code using the function **spdSetFieldNotificationFunction()**.
- Any time the field value is changed by other controls or through scripting code, the notification function is called, which will set the state of the check boxes to match the data. This includes a call to the function when the form is initialized so the check boxes will display the correct information when the form is first loaded.



RADIOBTN

The RADIOBTN example uses radio buttons to modify field data. The setup of a set of radio buttons is very similar to the check box example, except that each field requires multiple input controls, one for each radio button.

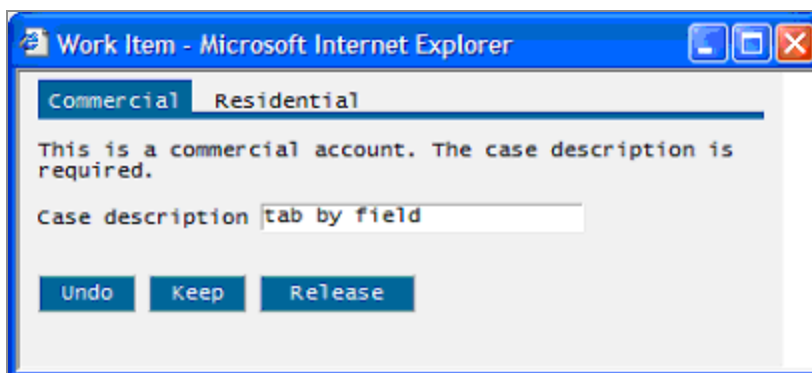


SCRIPT

The SCRIPT example lets the user enter a URL on the initial step. The form for the second step shows the URL as an embedded field. The web page for the URL is displayed on the form inside an IFRAME. A button will let the user alter the URL, which will both change the IFRAME to the new page and set the new value for the URL field.

TABFIELD

The TABFIELD example creates a set of tabs using HTML **** elements. Clicking on the tabs will change the value of a hidden field, which in turn causes different conditional sections of the form to be displayed.



TABNOFLD

The TABNOFLD example also creates a set of tabs, but it does not use an actual field value or rely on conditional sections of the form.

CSSCHGS

The CSSCHGS example shows changing the appearance of the iProcess Modeler Form through a **<style>** tag embedded in the form definition.

File Caching

The iProcess Modeler Form definition, including any embedded HTML, is converted into an HTML form for display in the iProcess Workspace (Browser).

The dynamically generated HTML can be further modified and saved to a specific path and file name on the server to be used for subsequent requests for the form. When this file-cached version of the form is detected, the iProcess Modeler Form definition is no longer used as a source for generating the HTML for the page.

To begin customizing a form, use the iProcess Workspace (Browser) to display a work item for the step you wish to change. Right click on the form and choose the option to view source. The source can then be modified and saved to the appropriate path and file name. Appropriate file names are listed in a comment near the top of the source.

In addition to modifying the section of the page used to display the iProcess Modeler Form, you may also modify header and footer sections of the page. You can then save the file to an additional location where it will be used as a source for the header and footer section for multiple steps or procedures. The comments at the top of the source indicate the appropriate file names for this purpose.

Setting up a Test Environment

Before saving your modifications to the location that will override the standard rendering of the form, you may want to save them into a work directory where you can fully test your changes before deploying. To do this:

1. Create a directory for storing the pages you will be modifying.
2. Copy the files `spddate.js`, `spdform.js`, `spdform.css`, and `memopage.html` from the directory where the Action Processor is installed to your work directory.
3. Open a work item in the iProcess Workspace (Browser) for the step you wish to customize.
4. Right click on the page for the work item and choose to view the source.
5. Save the source to your work directory. Any file name can be used in the work directory, but using the name you later plan to use in the file cache will make it easier to identify multiple files.

You can now load the saved page directly into your browser from the work directory. Everything should work as if you had just loaded the page from the iProcess Workspace (Browser), except that the **Undo**, **Keep**, and **Release** buttons will not actually communicate with the server. You can use any text or HTML editor to make modifications and test your changes.

Once you are satisfied with your changes, you can deploy by copying the file to the appropriate file name in the “htmlcache” subdirectory under the Action Processor installation directory (the htmlcache directory does not exist by default; you must create it).

Structure of the Complete iProcess Modeler Form Page

If you examine the source for a work item page returned by the iProcess Workspace (Browser), you will see a very simple skeletal web page with four areas clearly blocked off by start and end comments. Make sure you do not alter these start and end comments when you customize the page.

The four main areas are:

- Data XML
- Common Header HTML
- Common Footer HTML
- iProcess Modeler Form HTML

Data XML

This block is XML containing data for the work item / case that is displayed. The XML itself will not be visible on the page, but is embedded in the page as a source of information for the standard scripting inside of `spdform.js`. You should not write scripting code to directly query this XML because the format is subject to change in future versions. Only use the `spdform.js` functions to access the work item / case data.

After you have made some customizations to a form, you might want to try the form with data for a different work item. To do this, display a different work item in the iProcess Workspace (Browser), view the source, copy the Data XML section from that source, and use it to replace the same section in the copy of the page you are customizing.

Common Header HTML and Common Footer HTML

These are blocks of HTML that can be reused for multiple steps or procedures. These can be saved for use with a specific step, a specific procedure, or for all procedures. The information for the header and the footer are always retrieved from the same disk cache file so you can set up HTML in these sections that act as a wrapper for everything that lies between the header and footer. For instance, a table could be started in the header, a cell inside of the table could include what comes between the header and footer, then the footer would close that cell and specify the rest of the table.

Scripting code can be used to display data inside the header or footer. Make sure any fields you choose to display will always be available. For instance, fields like case description and case reference could be displayed for all procedures, but other case fields may not exist inside all procedures.

iProcess Modeler Form HTML

This is the block of HTML used for the iProcess Modeler Form. As with the header and footer sections, you can make modifications to the code and save the HTML to the cached HTML directory. You can choose to save it for use with a specific minor version of the procedure, for a specified major version of the procedure, or for all versions of the procedure. There are a few special requirements for this block of HTML, which will be described in the following sections.

Functions Available for File-Cached Scripting

The functions for embedded scripting are also available for file-cached scripting. For details about those functions, see [Functions Available for Embedded Scripting](#).

In addition, the following functions can also be used. Comments in the file **spdform.js** describe each function in detail.

Functions related to form initialization:

- **spdInitForm()**
- **spdAppendList**(listname, markingControlId)
- **spdInitMarking**(markingControlId, spdName, spdFormat, spdRequired, spdEmbedded, spdPreDecimalDigits, spdHelp)
- **spdCalculateOptionValue**(markingControlId, optionId, expression)
- **spdConditionalBlock**(blockId, condition)
- **spdPostInitForm()**

Near the end of the HTML that is dynamically generated for the iProcess Modeler Form is a block of script code that initializes the form. This should be left at the end since many of the function calls may reference text boxes, select lists, or other controls defined earlier in the form. The code begins with the **spdInitForm()** function and ends with the **spdPostInitForm()** function. The code in between performs actions such as populating

selection lists, initializing markings, and defining a function that updates the information on the form after a field is changed.

The **spdInitForm()** function sets up some hidden form elements that will be later used when submitting the form to undo, keep or release the work item.

When a marking has validation items, it will be rendered as an HTML **<select>** element. Individual selection items that are part of the form definition will appear as **<option>** elements under that. Items retrieved from lists maintained on the server, however, must be added through code. The **spdAppendList()** function retrieves the list values from the data XML block and adds them as **<option>** elements under the **<select>** element.

All markings are converted into HTML elements. In addition to **<select>** lists, these might be text boxes, hidden input areas, spans, or buttons. Calls to the **spdInitMarking()** function links up these controls to the appropriate fields and specifies how the data is formatted and whether it is required. When this function gets called during form initialization, the initial field value is retrieved from the data XML and placed into the HTML element that is defined.

Next, the **spdFormUpdate()** function must be defined. This function will be called whenever a field value changes. If any of the **<option>** elements for a **<select>** list are iProcess expressions (either a field name or a simple supported expression), there will be a call to the **spdCalculateOptionValue()** function to update the value for that **<option>** element.

If the form contains any conditional sections, there will be calls to the **spdConditionalBlock()** function to display or hide the section on the form.

Finally, the **spdPostInitForm()** function will be called. This makes an initial call to the **spdFormUpdate()** function and then performs an initial validation of the data in the form.

Functions related to marking data

- **spdSetMarkingValue**(markingControlId, fvalue)
- **spdGetMarkingValue**(markingControlId, getInitialValue)
- **spdIsMarkingValid**(markingControlId)
- **spdGetMarkingErrorMsg**(markingControlId)

These are similar to the functions for accessing field values, but rather than accessing the fields by name, it accesses them through the markings that have been set up on the form. When accessed this way, formatting information does not have to be specified, as that is defined for the marking when it is initialized.

For some dynamically generated controls, the **onchange** event will contain a call to the **spdSetMarkingValue()** function. This will validate the data in the field and notify any other markings for the same field of the change.

If the marking control is a text box, selection list or button, and it uses one of the standard class names to indicate it is an optional or required field, the colors of the text and background indicates the status of the data., as follows:

- Valid data - black on white
- Invalid data - white on bright red
- Required fields that are empty - dark red background
- Optional fields that are empty - dark blue background

If a custom notification is set up for the field, it is called. Also, the **spdFormUpdate()** function is called so that calculated validation items and conditional sections of the form can be properly updated.

Also, the **onkeypress** event for text boxes calls the **spdSetMarkingValue()** function, but will pass the optional parameter indicating that actions should only be performed if the last key pressed was the **Enter** key.

Functions related to form validation and submission

- **spdFormSubmit**(actionName)
- **spdSetValidationNotificationFunction**(notificationFunction)

In the dynamically generated HTML, the **Undo**, **Keep**, and **Release** buttons are defined between the area generated for the iProcess Modeler Form and the block of initialization scripting code. Each of these buttons calls the **spdFormSubmit()** function with the appropriate action name: “UndoForm”, “KeepForm”, or “ReleaseForm”.

You can set up a function that will be called after form validation is performed (done after any field change). The function should have two parameters. The first parameter will be true if the “KeepForm” action can be performed, and the second will be true if “ReleaseForm” can be performed.

During form validation, the **Keep** and **Release** buttons will be enabled or disabled as appropriate, and their appearance will be altered to indicate their enabled state: the **Keep** button is disabled if any fields contain invalid data, and the **Release** button is disabled if there is invalid data or an empty required field. To override this default behavior, you can

change the **id** attribute for the buttons and set up a validation notification function to trigger your own code.

Other Functions

- **spdEditMemo**(markingControlId, isReadOnly)
- **spdShowCalendar**(markingControlId, calendarLinkControl)
- **spdShowMarkingHelp**(markingControlId)

The **spdEditMemo**() function opens a separate dialog to edit or display a memo field. The memo will be editable unless you specify true for the optional second parameter.

The **spdShowCalendar**() function displays a calendar in the page. The *calendarLinkControl* parameter is required, and it should be set to an object that appears on the form. The calendar selection interface will be displayed next to the control specified. In the dynamically generated HTML, markings for date fields are followed by an **<a>** tag containing a small calendar selection graphic. This **<a>** tag is passed in as the calendar line control so the calendar appears to the right of that.

The **spdShowMarkingHelp**() function displays the help message defined for the marking in an alert box. The dynamically generated form only includes a link to display help if help text exists for that marking.

HTML for Marking Controls

This section describes the types of HTML controls that can be used as marking controls in a customized form. It describes each of the attributes of the control, values that can be assigned to those attributes, and the way they are typically set in the dynamically generated forms that will serve as the starting point for customizations.

All marking controls must have an **id** attribute, which uniquely identifies the marking control. This **id** is used to reference the control when initializing the marking, and in calls to other scripting functions. The name assigned in dynamically generated code is made up of the text “marking_”, plus the field name. If multiple markings for the same field are defined, a number is appended to keep the **id** unique.

A name attribute is assigned to each control, but is not actively used. The default name that is dynamically generated is the text “MARKING\$”, plus the field name.

Text Input Controls

Text input controls are implemented as an HTML **<input>** element, with the type of “text”.

The **class** attribute sets the general appearance. In the dynamically generated HTML, required markings will have a class of SPD_MARKING_REQ, optional markings will be SPD_MARKING_OPT, and display markings will be SPD_MARKING_READONLY.

When these standard class names are used, the text and background color of the control will change to reflect the status of the data (white on bright red if the data is invalid, black on white for valid data, a dark red background for blank required markings, and dark blue background for blank optional markings).

If a different class name is used, the field appearance will not automatically change based on the status of the data. Custom code could be added, using a field notification or validation notification function.

The **maxlength** attribute limits the length of the text that can be entered based on the field definition, and size will typically be set the same.

The **value** attribute should be left blank. When the marking is initialized, the value from the work item will be copied to the control.

The **onkeypress** and **onchange** events call the **spdSetMarkingValue()** function so that changes made in the text box will be applied to the work item data.

Selection Lists

Selection lists are implemented as an HTML **<select>** element. These are used for required and optional markings that have validation items.

The **class** attribute is typically set to SPD_MARKING_REQ for required fields and SPD_MARKING_OPT for optional. As with text boxes, the appearance of the control is automatically changed to indicate the validity of the data if one of these standard class names is used.

The **onchange** event will call the **spdSetMarkingValue()** function so that changes made to the selection will be applied to the work item data.

Embedded Markings

Embedded markings are implemented as an HTML **** element.

The **class** attribute of the **** tag is typically `SPD_MARKING_EMBEDDED`. This can be changed to alter the appearance of the marking without affecting functionality.

A parameter in the call to initialize the marking indicates whether the field data is embedded. When it is set to be embedded, the field text is wrapped in an HTML **<pre>** tag to preserve all white space. Multiple spaces and line feeds and certain characters are converted to an equivalent entity character to prevent any field text from being misinterpreted as HTML or XML.

Memo Markings

Memo markings are displayed as buttons, i.e., HTML **<input>** elements with a type attribute of “button”.

The **class** attribute sets the general appearance. Required memo markings will typically have a class of `SPD_MEMO_BUTTON_REQ`, optional memo markings will be `SPD_MEMO_BUTTON_OPT`, and display memo markings will be `SPD_MEMO_BUTTON_READONLY`.

The appearance of the buttons will change based on the data for the memo field when the standard class names are used, similar to text boxes and selection lists.

The **value** attribute for a button is, of course, used for the button text rather than data.

The **onclick** event for the **Memo** button calls the **spdEditMemo()** function, which displays a separate dialog window with the memo text. For read-only **Memo** buttons, an extra parameter is passed to this function to disable editing.

Calculated Markings

Calculated markings are rendered identically to display markings. They are shown as read-only text boxes.

Note that calculations are only performed when the work item is locked at the server and when the work item is kept or released at the server. If calculations need to occur in the browser, they will need to be implemented in custom scripting code.

Hidden Markings

Hidden markings are rendered as an HTML **<input>** element with the type “hidden” and simply store a copy of the current data for the marking field.

Standard Submit Buttons

The standard buttons for submitting the form use the following values for the **id** attribute.

- Undo - “undoButton”
- Keep - “keepButton”
- Release - “releaseButton”

The class for these buttons will typically be “SPD_BUTTON”. This can be changed without affecting functionality.

The **onclick** event for the button calls the **spdFormSubmit()** function with the appropriate action name.

When form validation is performed, the standard submit buttons are automatically enabled or disabled, depending on the state of the work item data. To override this behavior, change the **id** attribute to another value. Custom scripting code could then be used to change the state or appearance of the button, or the buttons could be left enabled at all times and the form validation routines will prevent submitting the form with invalid data.

Calendar Link

Text boxes for date fields will typically be followed by a link used to display a calendar for selecting a date.

In the dynamically generated HTML, this will be an HTML **<a>** tag with an **href** attribute set to “#” and an **onclick** event that calls the **spdShowCalendar()** function. Inside the link is a **** element with a **class** attribute of “SPD_CALENDAR” that sets a background image for the span, which is a calendar selection icon.

This link could be replaced with any HTML for calling the **spdShowCalendar()** function.

Help Link

Text boxes that have associated help text will typically be followed by a link for displaying the help text.

In the dynamically generated HTML, this will be an HTML **<a>** tag with an **href** attribute set to “#” and an **onclick** event that calls the **spdShowMarkingHelp()** function. Inside the link is a **** element with a **class** attribute of “SPD_HELP” that sets a background image for the span, which is a help icon.

This link could be replaced with any HTML for calling the **spdShowMarkingHelp()** function.

Field and Form Validation

There are three locations where you might insert custom scripting code to react to a change in a field value.

- A field notification function, if one has been defined for the changed field.

The field notification is typically set up when something other than a marking is used to modify the data (e.g., editing data through check box or radio buttons). Although the function could also be used to trigger actions for one of the supported marking controls.

- The **spdFormUpdate()** function, which is required as part of the scripting code for the form.

The **spdFormUpdate()** function typically updates the options for selection lists when those are reference field values or are calculated from a simple expression. Also, it is used to hide or display conditional sections of the form. Custom code could also be added to the function.

- A validation notification function, if one has been defined for the form.

The validation notification function is called last, after the default form validations are performed to determine whether the work item can be kept or released. Custom code here can override the status determined for keep and release, or to perform any other actions.

The validation notification function is also called right before the form is submitted. So even if the function doesn't disable every method of keeping or releasing the work item, it can still block the actual keep or release. When the form submit is blocked in this way, an alert box is displayed explaining why.

File-Cached Customization Example

A file-cached customization example is provided in the following directory:

```
InstallationHomeDir\iprocessorclientbrowser\samples\  
IPCBrowserFormExamples
```

where *InstallationHomeDir* is the directory in which the installer places administrative files, such as the uninstaller, documentation, and sample code. This defaults to `C:\tibco` on Windows systems, and `/opt/tibco` on UNIX systems, but can be specified as a different directory when the iProcess Workspace (Browser) is installed.

This example includes a `IPCBrowserExamples.xpd1` file, which contains a sample procedure named `SITERATE`. It contains three steps.

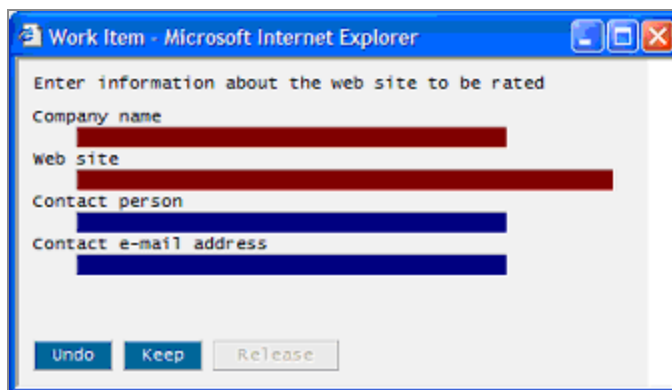
- `INIT` - Lets a user enter the basic site information for the site that will be rated later.
- `RATE` - Lets a reviewer rate the site and provide comments.
- `FINAL` - Displays a report on the rating.

The example also includes the following files, which customize the procedure for display in the iProcess Workspace (Browser). To begin using these customizations, copy these files into a subdirectory named “`htmlcache`” under the Action Processor installation directory.

- `common_SITERATE.html`
- `SITERATE_FINAL.html`
- `SITERATE_INIT.html`
- `SITERATE_RATE.html`

The following screen shots show the steps of the procedure without the customizations.

STEP 1: Initial form for data entry



Work Item - Microsoft Internet Explorer

Enter information about the web site to be rated

Company name

Web site

Contact person

Contact e-mail address

Undo Keep Release

STEP 2: Form for rating the web site

Work Item - Microsoft Internet Explorer

Company Name: Sample Company
Company Web Site: www.sample.com

Web site rating. score the site in the following areas (1-lowest, 5-highest).

Navigation	4
Layout	4
Content	3
Search	2
Overall	

General Comments

Identify problem areas detected on the web site.

Missing contact or feedback links	Yes v
Invalid or broken links	No v
Outdated or invalid content	Yes v
Missing search functionality	No v
Other (please describe)	No v

Comments on problems

Reviewed by Reviewer 31

STEP 3: Final report on the web site rating

Work Item - Microsoft Internet Explorer

Report on web site rating - reviewed by Reviewer 31

Company Name: Sample Company
Company Web Site: www.sample.com
Contact Person: Sample person
Contact email: sample@sample.com

Overall Site rating: 3.2

Navigation	4
Layout	4
Content	3
Search	2

=== Overall site comments ===
Most aspects of this site are well thought designed. I was not able though to submit information for correcting information about some of the companies older products.

=== Problem areas ===
The site was missing contact or feedback links.
The site contained outdated or invalid content.
Several older products missing information about system requirements.

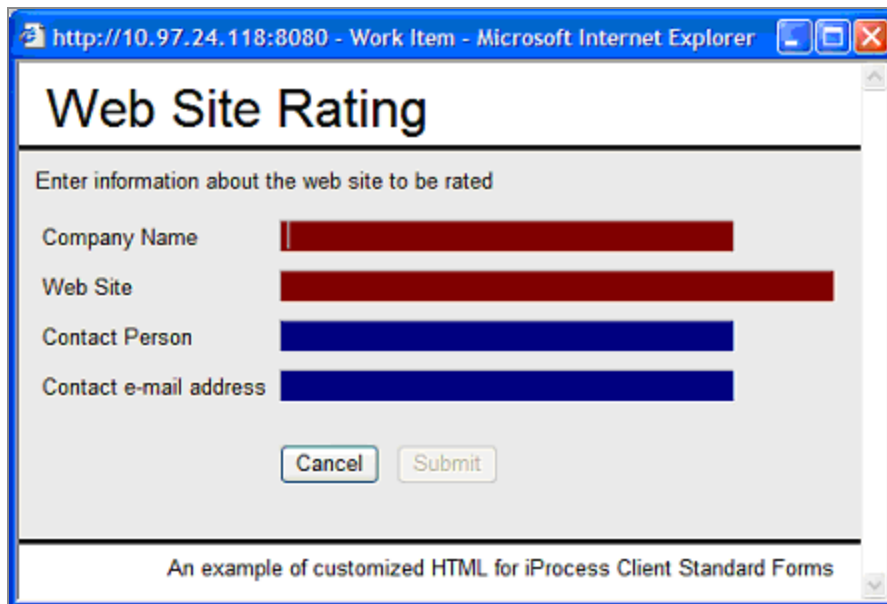
The customized version performs a wide variety of customizations. For example:

- Some of the data entry controls, with sets of radio buttons and check boxes.
- Proportionally spaced fonts are used for all text.
- Custom colors for all elements.

- Tables are used for positioning text and input controls.
- Hyperlinks rather than buttons are used to trigger editing of the memo fields.
- The memo data was added to the page as an embedded marking.
- The web site address is made into a hyperlink that loads the page to be rated in a separate browser window.
- The submit buttons appear as standard browser buttons.
- The **Keep** button is omitted on forms where it isn't needed.
- Text for the submit buttons is changed to fit the process rather than using generic terms.
- A file with header and footer HTML is used to define static content used for all steps.

Here are the customized forms for these same steps:

STEP 1: Initial form for data entry



The screenshot shows a Microsoft Internet Explorer window with the address bar displaying 'http://10.97.24.118:8080 - Work Item - Microsoft Internet Explorer'. The main content area is titled 'Web Site Rating' and contains the instruction 'Enter information about the web site to be rated'. Below this, there are four input fields: 'Company Name', 'Web Site', 'Contact Person', and 'Contact e-mail address'. The 'Company Name' and 'Web Site' fields have red text, while the 'Contact Person' and 'Contact e-mail address' fields have blue text. At the bottom of the form are 'Cancel' and 'Submit' buttons. A footer note at the bottom of the browser window reads 'An example of customized HTML for iProcess Client Standard Forms'.

STEP 2: Form for rating the web site

Work Item - Microsoft Internet Explorer

Web Site Rating

Company Name

Web Site

Web site rating. Score the site in the following areas on a scale of 1 to 5 (1-lowest, 5-highest).

Navigation	☐ 1	☐ 2	☒ 3	☐ 4	☐ 5
Layout	☐ 1	☐ 2	☐ 3	☒ 4	☐ 5
Content	☐ 1	☐ 2	☐ 3	☐ 4	☒ 5
Search	☐ 1	☐ 2	☒ 3	☐ 4	☐ 5

Overall rating

General Comments [edit](#)

This site contains excellent articles and product information. The search facilities make it fairly easy to find the appropriate information.

However, the many of the standard navigation links lead to blank pages or 'resource not found' message although the appropriate page can be found through search.

Problems Areas

- ☐ Missing contact or feedback links
- ☒ Invalid or broken links
- ☐ Outdated or invalid content
- ☐ Missing search functionality
- ☒ Other (please describe)

Problem Area Comments [edit](#)

The narrow font used for headers on many pages is very difficult to read.

Reviewed by

An example of customized HTML for iProcess Client Standard Forms

STEP3: Final report on the web site rating

Work Item - Microsoft Internet Explorer

Web Site Rating

Company Name	Sample Company 2
Web Site	http://www.sample2.com
Contact Person	Sample Person
Contact email	sample@sample2.com
Reviewed by	Reviewer 14
Overall Site Rating	3.7
Navigation	3.0
Layout	4.0
Content	5.0
Search	3.0
Overall site comments	This site contains excellent articles and product information. The search facilities make it fairly easy to find the appropriate information. However, the many of the standard navigation links lead to blank pages or 'resource not found' message although the appropriate page can be found through search.
Problem Areas	The site contained invalid or broken links. The narrow font used for headers on many pages is very difficult to read.

Close Eliminate from listings

An example of customized HTML for iProcess Client Standard Forms

Common Issues for Embedded and File-Cached Customizations

You may wish to reference images, JS files, CSS files, or other external files in your custom HTML code. If you will be using a relative path to access these files, keep in mind that all paths are relative to the Action Processor installation directory.

Although the file cache for your modified HTML is the subdirectory “htmlcache”, underneath the Action Processor directory, this is not the base directory for pages generated using those files as a source for the iProcess Modeler Form or form header and footer sections for the page.

Therefore, if you store images in the directory “htmlcache\images”, you will need to use “htmlcache\images” as the relative path to those files in either embedded or file-cached HTML.

When using a working directory to create and test your file-cached HTML, you will need to create a copy of any files referenced by relative path to an appropriate directory under your working directory.

Displaying Forms Outside of the iProcess Workspace (Browser)

This section describes an example that is provided in the TIBCO iProcess Workspace (Browser) that allows you to display iProcess Modeler-created forms outside of the TIBCO iProcess Workspace (Browser).

The LinkForm Example

An example is provided with the TIBCO iProcess Workspace (Browser) that allows you to display iProcess Modeler-created forms outside of the TIBCO iProcess Workspace (Browser). (This example is applicable only to iProcess Modeler-produced forms.)

The `LinkForm.html` file can be found in the installation home directory:

```
InstallationHomeDir\iprocessclientbrowser\samples\LinkFormExample
```

where *InstallationHomeDir* is the directory in which the installer places administrative files, such as the uninstaller, documentation, and sample code. This defaults to `C:\tibco` on Windows systems, and `/opt/tibco` on UNIX systems, but can be specified as a different directory when the iProcess Workspace (Browser) is installed.

The `LinkForm.html` file is an example of an intermediary HTML file that can be used to display an existing work item without running the TIBCO iProcess Workspace (Browser). Note that this can only be used to display standard iProcess Modeler-produced forms. Those forms can include customizations made by embedding HTML into the iProcess form definition or made by creating a file-cached copy of the HTML as described in [Customizing iProcess Modeler Forms](#). Custom GI Forms, Form Flow forms, or other external form systems that may be implemented in the future cannot be displayed in this manner.

Also note that this is a simple HTML example that passes login information directly through a URL and performs manipulation of the request through client-side scripting. As such, it does not represent a best practice solution regarding security. However, the general ideas presented can be used in building a custom Servlet, JSP, or ASPX page.

The parameters passed in a link to the `LinkForm.html` file are parsed through scripting code. The following parameters are supported in the example:

- `workitemtag`
- `nodename`
- `computername`
- `ipaddress`
- `tcpport`
- `isdirector`
- `nodealias`
- `username`
- `password`

where:

- **workitemtag** identifies the work item. This is required.
- **nodename**, **computername**, **ipaddress**, **tcpport**, and **isdirector** identify the TIBCO iProcess Objects Server. Either these parameters can be provided to identify the server, or you can provide the **nodealias** (see below).
- **nodealias** identifies the TIBCO iProcess Objects Server. Either this parameter can be provided to identify the server, or you can provide the five parameters listed above (**nodename**, **computername**, etc.). If a node alias name is specified, code inside of the **LinkForm.html** must set the other five values to identify the iProcess Objects Server.
- **username** and **password** provide login credentials. These are optional. If both are provided, the work item will immediately be displayed. If either of these is omitted, a login section of the page will be displayed allowing the user to enter the information. If there is a problem logging in with the information provided, a message is displayed and it will return to the login interface. (Note that all parameters are passed to the `LinkForm.html` file as part of the URL, so specifying the password in the **password** parameter is not advised.)

The URL should be launched using the javascript **window.open(...)** command, so the work item will appear in a separate window without browser menus or toolbars. The following is an example of this command:

```

window.open
('http://myserver/actionprocessor/linkform.html?workitemtag=myserver|TEST
PROC|swadmin|R|4475|19048|myserver|FORM1|0|7@nodealias=myserver@username=
swadmin', '_blank', 'resizable=1,scrollbars=1');

```

The following is an example of HTML for a hyperlink displaying the work item:

```

<a target="_blank" href="" onclick="window.open
('http://myserver/actionprocessor/linkform.html?workitemtag=myserver|TEST
PROC|swadmin|R|4475|19048|myserver|FORM1|0|7@nodealias=myserver@username=
swadmin', '_blank', 'resizable=1,scrollbars=1');return false;");">Display
work item</a>

```

To use the **LinkForm** example:

1. Examine the contents of the LinkForm.html file for “TODO” messages and make the appropriate changes. The “TODO” items are for the following:
 - Specifying the appropriate Action Processor name, which will be different for Java and .NET versions.
 - Defining node aliases, which will allow you to pass fewer parameters in the URL.
2. Save the LinkForm.html file in the Action Processor installation directory.

Using Forms with User Tasks

This section explains how to work with forms associated with a user task of a business process and used in the context of TIBCO iProcess Workspace (Browser) or TIBCO iProcess Workspace (Browser) Components.

Refer to *TIBCO Business Studio Forms User's Guide* for general information about designing forms.

Overview

The information in this section is useful if you are responsible for designing and implementing business processes that include user interfaces for presenting and capturing information from users. The forms and business processes you design are deployed to the iProcess server using TIBCO Business Studio. The business process and associated forms are accessed using TIBCO iProcess Workspace (Browser).

Creating a New Form

When you create a new business process, a default form is created for each user task defined in the business process. You can use the default form or if required, edit the default form. You can also use either of the approaches listed below to create a new form for a user task:

- [Creating a New Form for an Existing User Task](#)
- [Creating a New Form Manually from the Project Explorer](#)
- [Linking a Free-Standing Form to a User Task](#)
- If required, [Switching back to the Default Form](#)

Creating a New Form for an Existing User Task

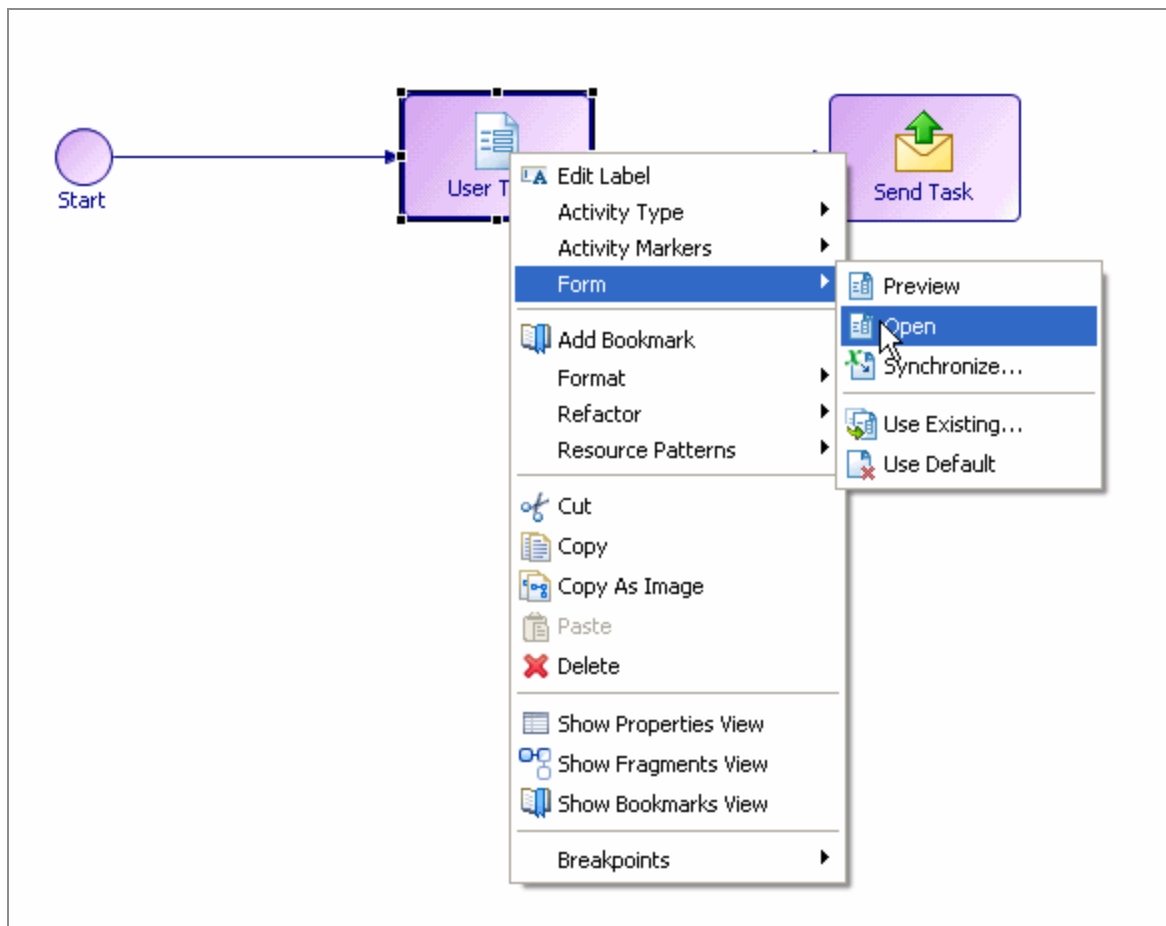
The most efficient way of developing usable forms is to edit the default form for an existing user task. Following are the reasons:

- From the beginning, the default form exists in the context of the user task with which it is associated in the business process. This means that the **Form** field for the user task (on the **General** tab of the user task's Properties view) is automatically filled in with the URL of the new form.
- If the user task has parameters associated with it, the new form includes user interface components, bound to the corresponding form parameters.

You can create the default form for any user task in TIBCO Business Studio in one of the following ways:

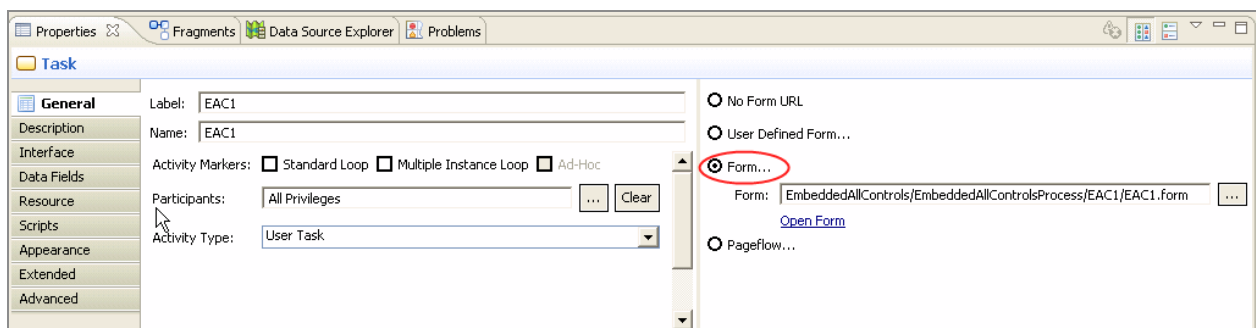
- Navigate to the context menu of a user task in a business process and click **Form > Open**.

Figure 1: Form > Open Option for a User Task



- Navigate to the Properties view of a user task, and on the **General** tab, select the **Form** radio button.

Figure 2: General Properties Tab of the User Task



In both of these cases, the existing default form is set on the user task as the custom form. You are prompted with a warning message that reads, "the customised form will no longer be automatically kept in sync with the activity interface."

Now, use the automatically created default form as a starting point for further modifications. Any changes to the user task interface or its associated data types must be synchronized with the form. The form displays problem markers when the parameters defined for the form are out of sync with the activity interface from which it was created.

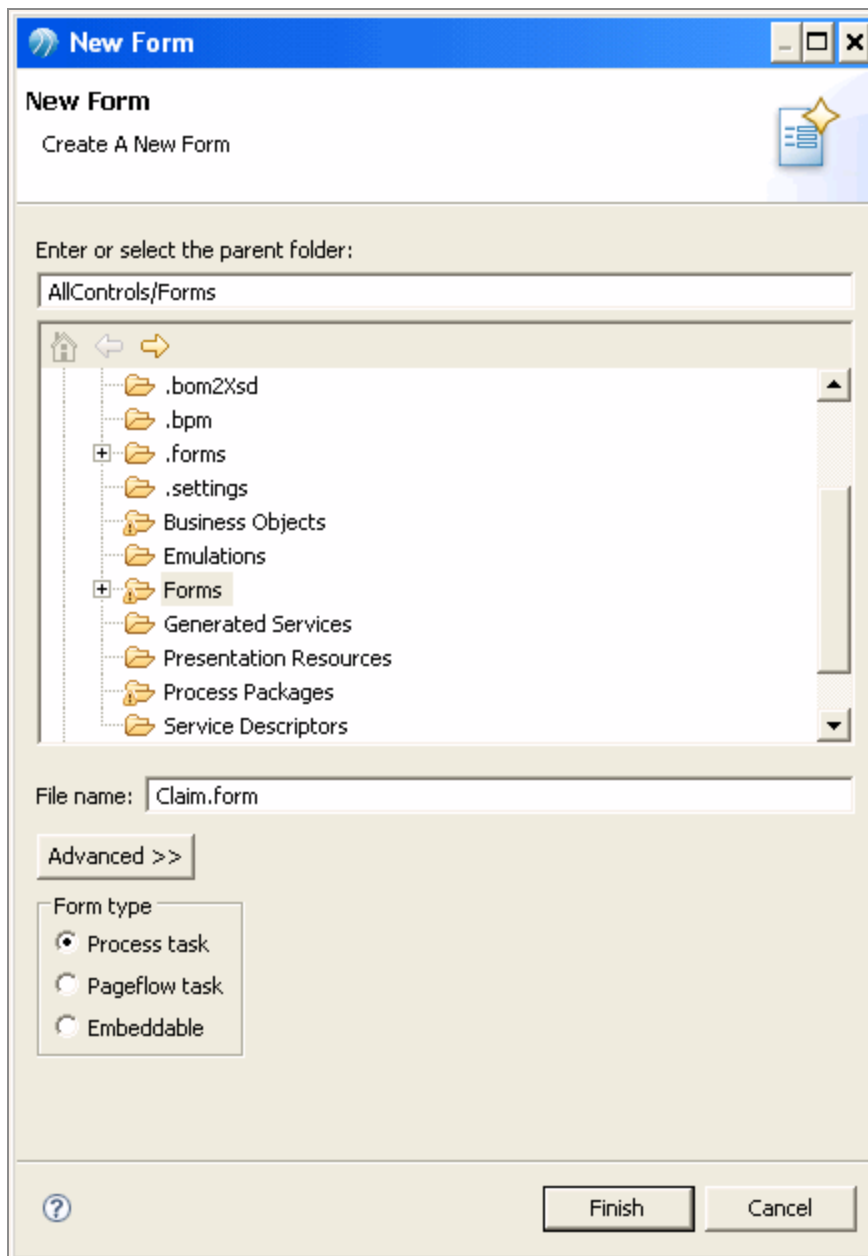
Creating a New Form Manually from the Project Explorer

You can create a form manually from the Project Explorer within any **Forms** special folder using either of the following approaches:

- Go to the context menu of the **Forms** special folder, or any child folder under the **Forms** special folder in the Project Explorer, and click **New > Form**.
- On the **File** menu, click **New > Other > TIBCO Forms > Form**.

This triggers the opening of the **New Form** dialog as shown in [New Form Dialog](#).

Figure 3: New Form Dialog



Specify the **Form type** on the **New Form** dialog. The type of form that is selected here determines the components that are initially part of the form model. The relevant form types are as follows:

- **Process task:** This creates a form identical to the one created from a User Task in a process definition. It contains a root pane, a toolbar with **Cancel**, **Close**, and **Submit** buttons, and a message pane for displaying error messages.

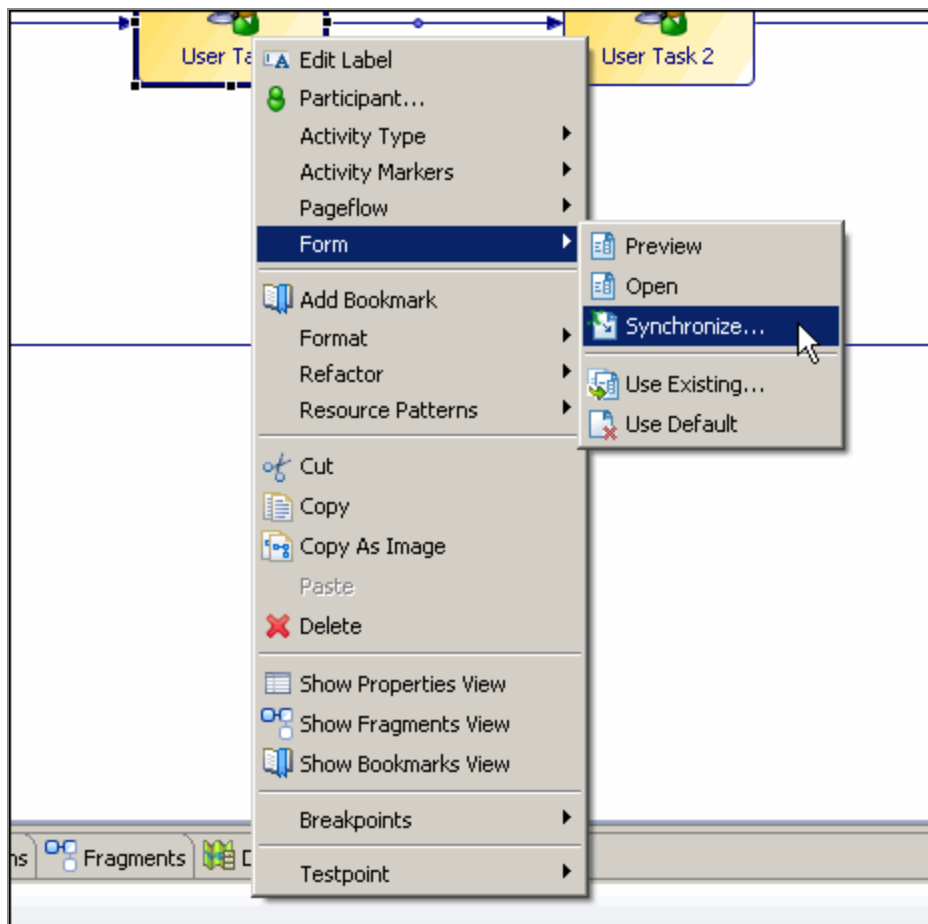
- **Pageflow task:** This creates a form identical to the one created from a User Task in a Pageflow process. The only difference to a Process task form is that the toolbar contains only **Cancel** and **Submit** buttons. The **Close** operation is not supported in pageflows because there is no way to reopen a step in a pageflow once it is closed.
- **Embeddable:** This creates a form that is suitable for embedding within another form. This contains only a single root pane. This is because the parent form would typically contain the toolbar and messages pane, so these components are not needed in an embeddable form. You can refer to *TIBCO Business Studio Forms User's Guide* for more information about Embeddable Forms.

Linking a Free-Standing Form to a User Task

You can use stand-alone forms for existing user tasks.

1. In the context menu of the user task, click **Form > Use Existing**.
The **Select Form** dialog opens.
2. Select an existing form from the Project Explorer folder structure and click **OK**.
Note that you must choose a form within the same project as the user task.
3. Synchronize the user task parameters with the form by going to the context menu of the user task and clicking **Form > Synchronize**. This establishes a permanent link from the form to the user task. This also enables the form synchronization validator to check that the form remains in sync with the user task's data interface and for the form synchronization wizard to modify the form as necessary to preserve consistency.

Figure 4: Synchronize the User Task Parameters



Note

The bidirectional linkage between the user task and form means that any given form may only be used by a single user task. The pairing of a user task with a form can be subject to automatic synchronization validation. If you share a form between multiple user tasks, you must manually synchronize the form with the different user tasks. By doing this, the synchronization validation link to the previously associated user task breaks. Hence the second approach is not recommended. A better way to share forms is by embedding the reusable parts.

Warning

Forms must be created only under a **Forms** special folder, or within the folders under a Forms special folder. If you choose a location other than the default location, be sure that the folder you select meets this requirement.

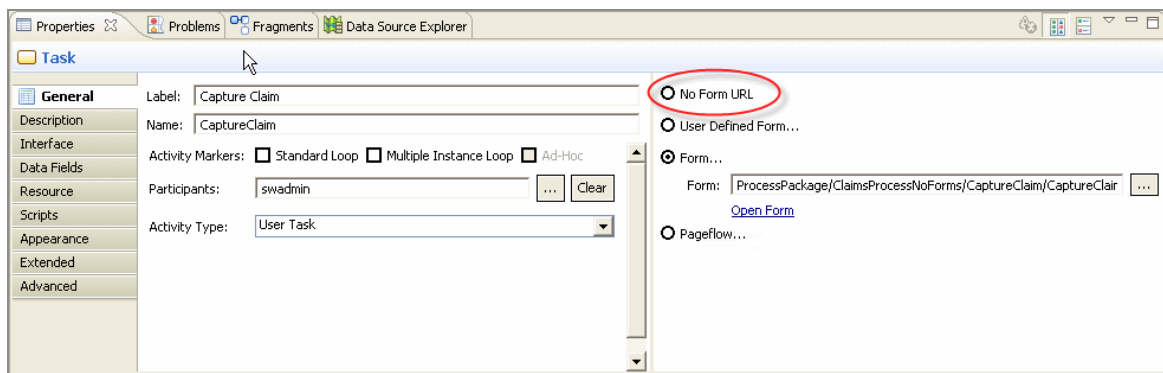
Switching back to the Default Form

After creating an auto-generated form using the **Form > Open** context menu action or using the **Form** radio button in the General Properties tab of the user task and manually customising the form, there can be a situation where you may want to start over again with a default form.

To re-create a default form, do the following:

1. In the context menu of the user task, click **Form > Use Default**. You can also select the **No Form URL** radio button in the General properties tab of the user task as shown in [General Properties Tab of the User Task](#).

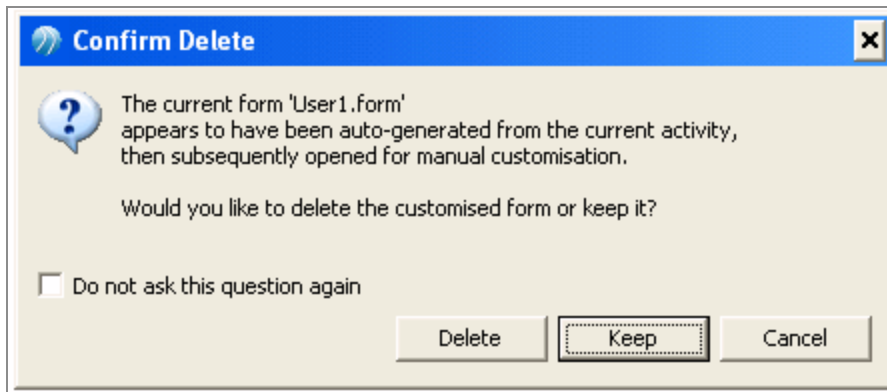
Figure 5: General Properties Tab of the User Task



The **Form > Use Default** option is enabled only if you select the **Form** radio button on the General tab of the user task Properties view.

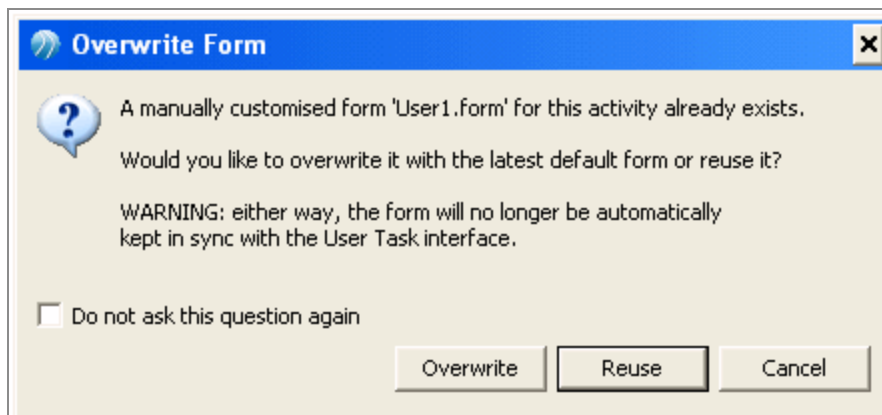
2. The **Confirm Delete** dialog appears.

Figure 6: Confirm Delete Dialog



3. Select **Delete** to delete the existing customised form. The customised form in the Forms folder in the Project Explorer is deleted.
4. Select **Keep** to retain the customised form. You may want to retain the form if it were still to be used by other user tasks.
5. After providing the delete options, generate the new form. In the context menu of the user task, click **Form > Open**.
6. If you previously specified the **Delete** option on the **Confirm Delete** dialog, a default auto-generated form is created.
7. If you previously specified the **Keep** option on the **Confirm Delete** dialog, the **Overwrite Form** dialog displays.

Figure 7: Overwrite Form Dialog



8. Select **Overwrite** to overwrite the existing customised form with a default form or select **Reuse**, to continue using the existing customised form.

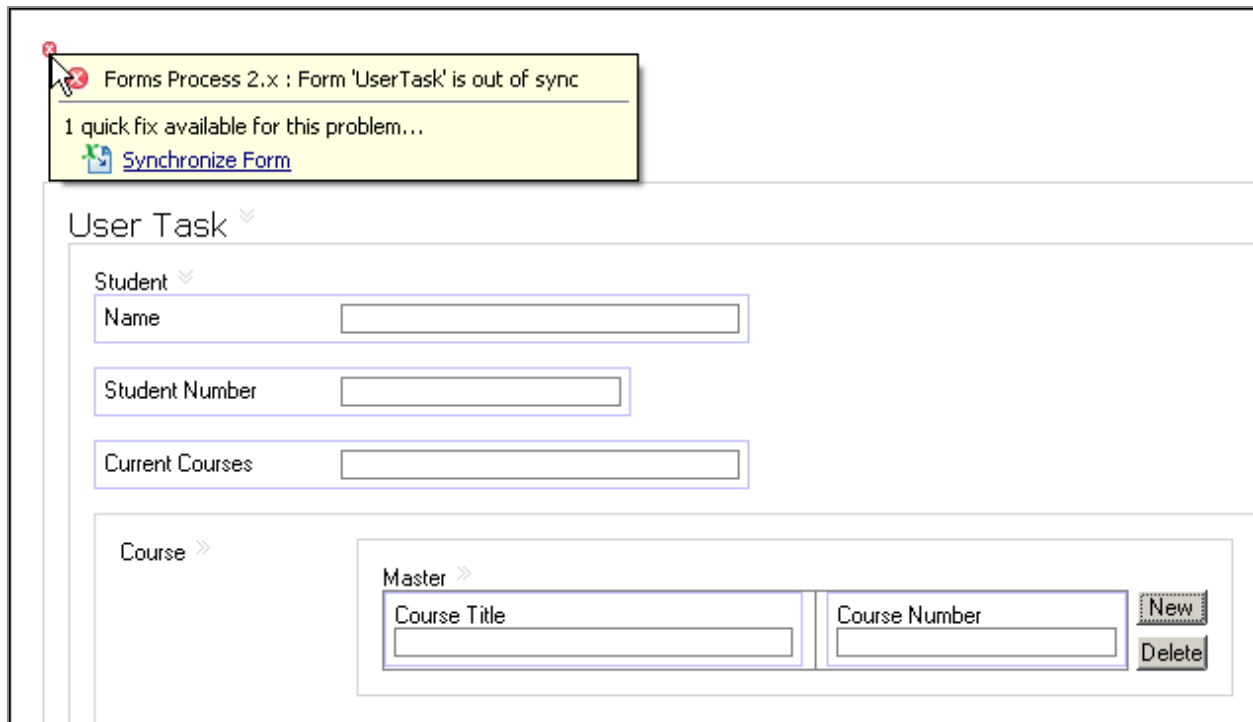
9. The system saves your responses. Your previous choice becomes the default action (The action executed if you press the Enter key). If you select the **Do not ask this question again** check box, the dialog is not displayed on subsequent occasions. The system automatically takes the action that you selected on this occasion. These settings may be changed using **Window > Preferences > Form Designer > Process**.

Updating Forms with Synchronization Wizard

If the set of task parameters changes after a form is created, you can use the **Synchronize Form** wizard to update the set of form parameters, controls, and bindings in the linked form. Likewise, if there is a change in the business object model that impacts the form (for instance, if an attribute is added to a class used by the form), the synchronization wizard updates the form appropriately.

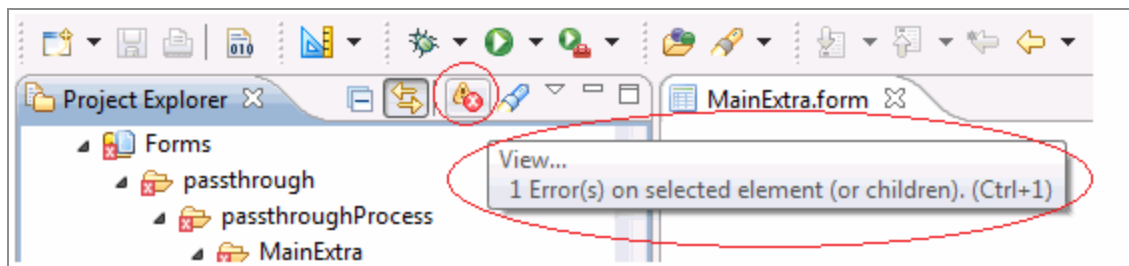
When a form needs to be synchronized, a problem marker decoration appears on the form in the Form Designer, usually near the element that is out of sync, as well as in the Properties and Content Outline views. You can see the problem marker in the **Problems** view. Move the cursor over the problem marker on the form to see a tool tip containing a description of the problem and a link to launch the synchronizer, as shown in [Problem Marker: Form out of sync](#).

Figure 8: Problem Marker: Form out of sync



You can also use the  button available in the Project Explorer view, Outline view, and Properties view. When the problem markers are in scope for the input of the active view, this action is visible. You can click it to inspect the markers and execute quick fixes.

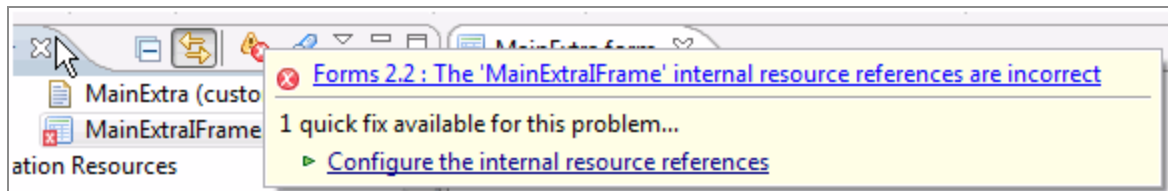
Figure 9: Problem Marker in the Project Explorer View



Click the  button, to display the following two hyperlinks:

- The marker link (opens the offending resource in its appropriate editor)
- The quick fix link (applies the quick fix)

Figure 10: Problem Marker in the Outline View



If there are multiple markers, the tooltip window displays left and right arrow buttons to navigate the markers. If a marker issue has multiple quick fixes, they are all listed in the lower section.

Options for Synchronization

If quick fixes are available for several components on the form, the tool tip includes left and right arrows for navigating through them. Also, in the case of multiple synchronization problems, the tool tip presents up to three different options for handling the problems.

You can either synchronize the entire form, or synchronize just the component whose problem marker was used to launch the wizard. In the latter case, although fixes for all out-of-sync components are displayed, *only* the component whose problem marker you moved the mouse cursor over has the check boxes selected for its available synchronization actions. That is, only that component is selected for updating by the wizard.

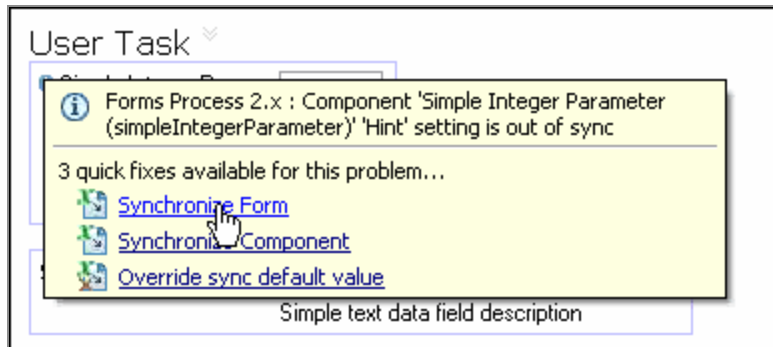
The third option when multiple components are out of sync is to override the sync default value. That is, to leave the form as it is. This may be because the problem is minor or because you intend to fix the user task parameter instead, to bring the user task and the form into sync. In this option, the wizard ignores the suggested fix or fixes, and removes the problem marker for the component whose marker was used to launch the wizard.

i Note

If the wizard is subsequently launched with the **Synchronize component** option using the problem marker of another component, fixes for the component for which the override was done are still displayed, but are read-only to indicate the override. Nonetheless, the read-only fixes can be selected, undoing the earlier override. In this case, the wizard displays a warning message that user overrides will be overwritten with generator defaults.

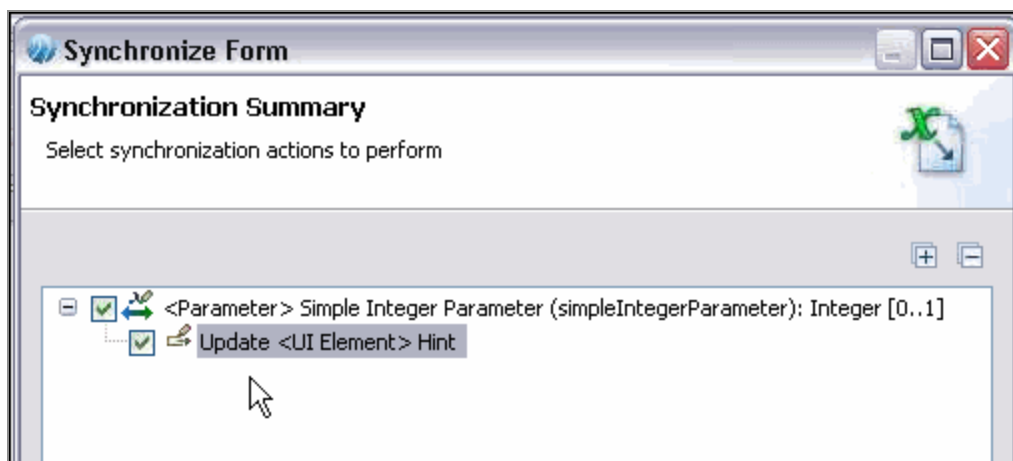
[Options for Synchronization](#) shows an example of the three possible options for synchronization presented by the tool tip: synchronize form, synchronize component, and override sync default value.

Figure 11: Options for Synchronization



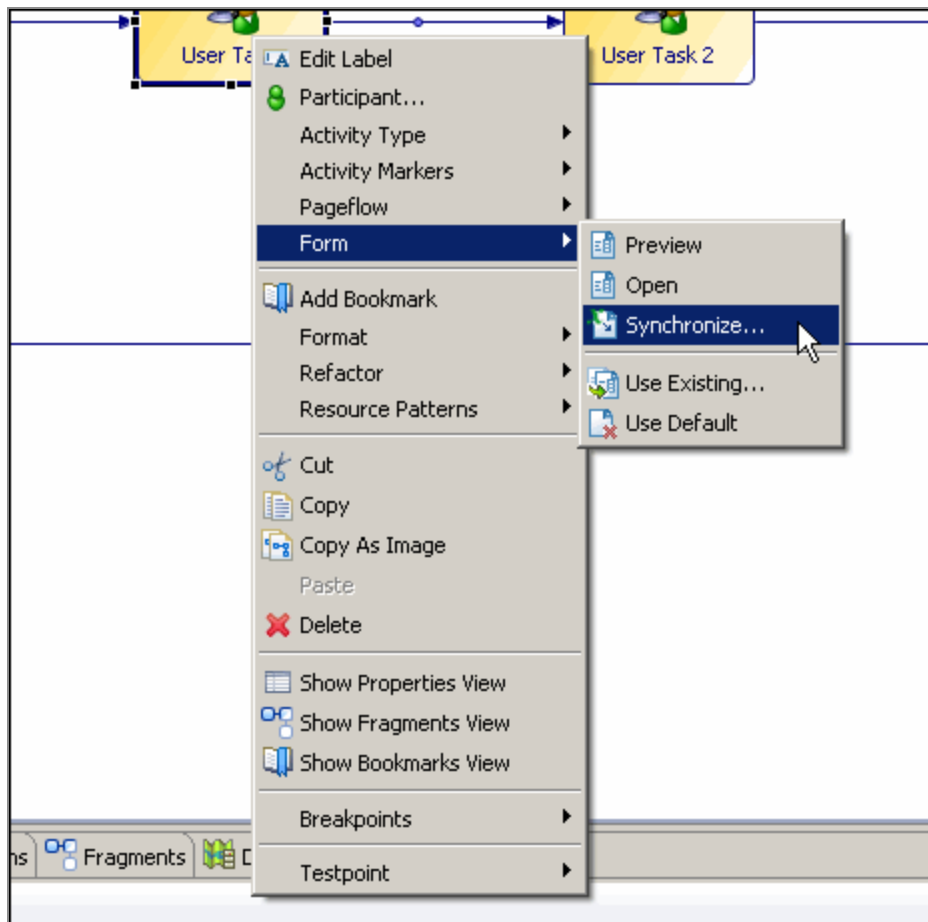
Click the desired link in the tool tip to launch the **Synchronize Form** wizard.

Figure 12: Synchronization Summary Page



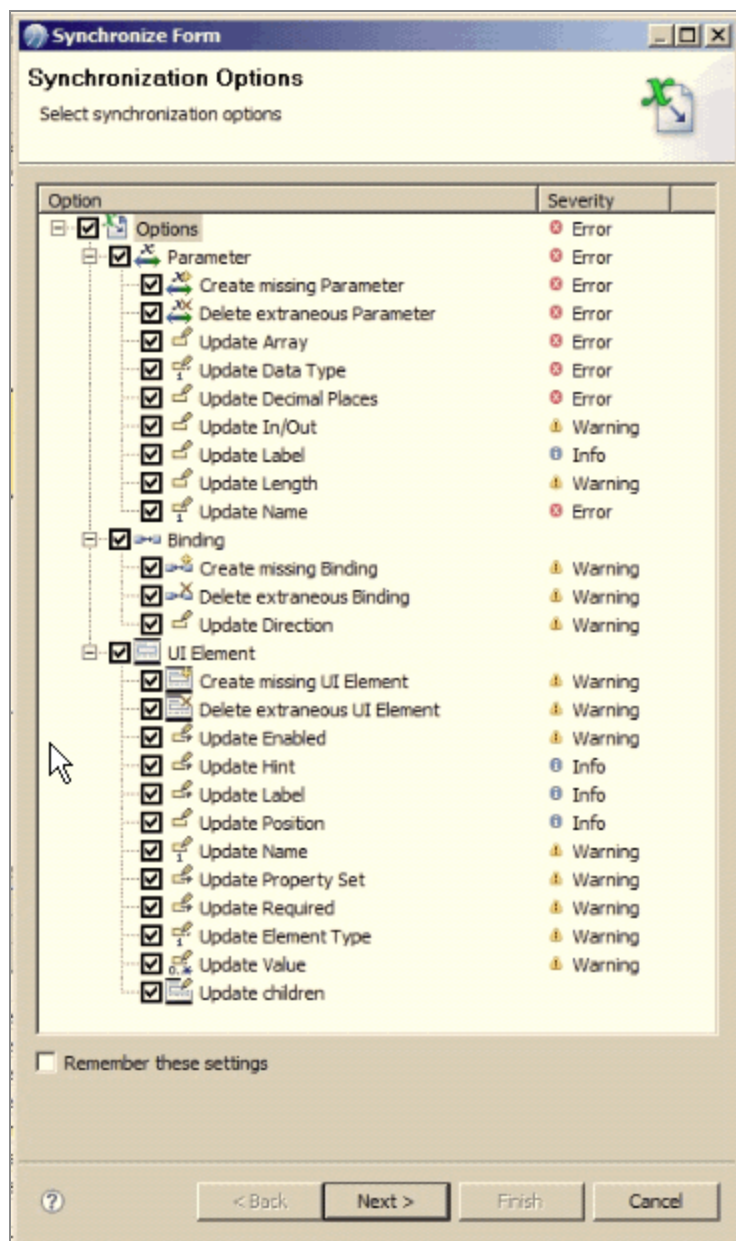
To launch the wizard directly, without using a tool tip, right-click the user task, and click **Form > Synchronize**. The menu for launching the wizard is shown in [Menu to Launch the Wizard to Synchronize Parameters in the Form](#).

Figure 13: Menu to Launch the Wizard to Synchronize Parameters in the Form



The first screen of the wizard, shown in [First Page of Synchronization Wizard](#), presents a list of all the possible conditions on which the wizard can check and report, with the severity associated with the out-of-synch item. Normally, all of the check boxes can be left selected.

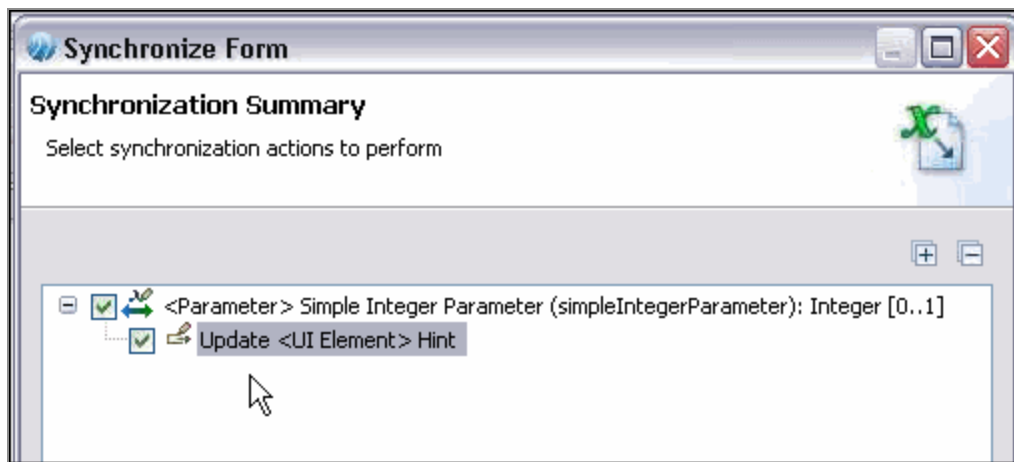
Figure 14: First Page of Synchronization Wizard



If you select the **Remember these settings** check box, the wizard does not start on page one, **Synchronization Options**, but starts on page two, **Synchronization Summary**. From page two, you can still navigate to page one by clicking the **Back** button.

Click **Next** to advance to the **Synchronization Summary** page, which displays the changes that can be made, such as parameters to be added, modified, or removed from the form. Clicking the Expand All (+) or Collapse All (-) buttons on this page expands or collapses the list of actions displayed. Using the check boxes, you can select the actions the wizard performs when you click the **Finish** button.

Figure 15: Synchronization Summary Page



Actions performed by the wizard respond to the **Undo** and **Redo** commands invoked on the **Edit** menu. Note that if a synchronization fix is undone, recreating the out-of-sync condition, the problem marker may not be displayed again until the form is saved or the synchronization wizard is invoked again.

Synchronization Wizard Preferences

By editing the **Synchronization Options**, you can modify the list of conditions that are selected by the wizard (shown on the first page of the wizard), and the severity level indicated by the problem markers (and displayed in the tool tip) for each item. In the menu bar, click **Window > Preferences > Form Designer > Process** to view and edit the synchronization settings. You can also click **Window > Preferences > Form Designer > Process** to open the dialog and select or clear the check boxes to remove or add an item to the list of items that is selected when the wizard is run. Click the severity for an item to display an option list for selecting a different severity. In most cases, the default severity choices are appropriate.

You can also click **Window > Preferences > Form Designer > Errors/Warnings > Process Synchronization** to view the following:

- View descriptions of the items that can be synchronized by the wizard
- View or edit the setting for the severity associated with an out-of-sync condition for each item

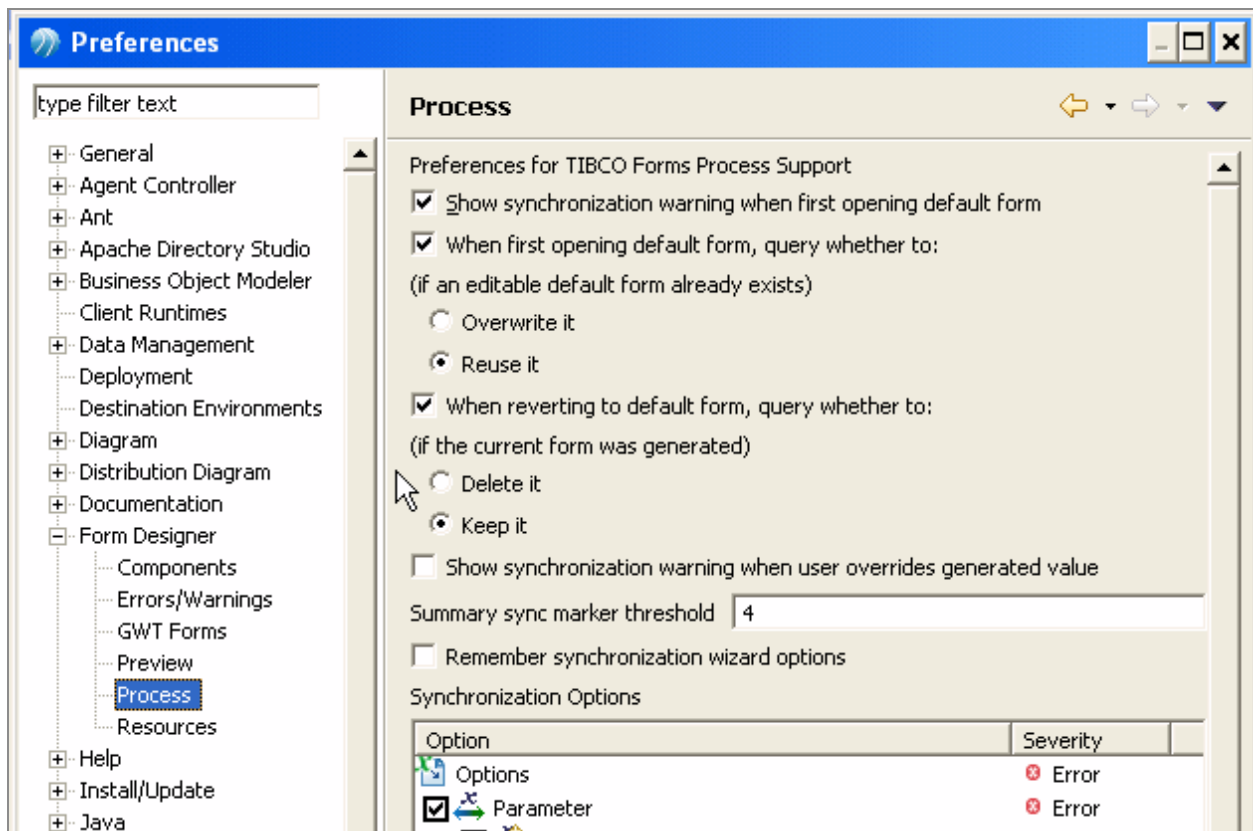


Warning

The presence of an **Error** marker prevents deployment of the form, while **Warning** and **Info** markers do not. If severity of the marker for an item is changed from **Error** to **Warning** or **Info**, the form is deployable, but doing so is strongly discouraged because the severity level of **Error** indicates a likelihood of severe problems with the form at runtime if the condition is not addressed. For this reason, normally leave the severity levels unchanged.

The additional preferences affecting synchronization can be edited in the Process dialog (**Window > Preferences > Form Designer > Process**), shown in [Preferences for TIBCO Forms Process Support](#).

Figure 16: Preferences for TIBCO Forms Process Support



Most of the settings here are self-explanatory. The **Summary sync marker threshold** setting deserves a mention, however. This is the number of sync markers on the form above which individual problem markers are not shown. By default, the threshold is 4. Change this setting by typing the desired number in the text field. If the number of sync

markers on the form are above the **Summary sync marker threshold** value, only one summary marker reading "Form <form name> is out of sync," is displayed.

If you select the **Remember synchronization wizard options** check box, the wizard does not start on page one, **Synchronization Options**, but starts on page two, **Synchronization Summary**. From page two, you can still navigate to page one by clicking the **Back** button.

Using Data Fields and Parameters with Process User Tasks

Data fields and process parameters represent data that can be used as an input to or output from steps in a business process. When mapped to a user task, process parameters represent the inputs and outputs of the user task. The parameters of a user task, in turn, can be mapped to controls on the form associated with that user task.



Note

While XPDL defines two types of parameters: `FormalParameter` and `ActualParameter`, the term parameter is used in TIBCO Business Studio as follows:

Formal ParametersParameters created at the process level of a process package represent interfaces for other processes that want to provide input to or receive output from the process that contains these parameters. They can be used as an input to or output from any user task in the business process if they are added to the set of parameters of a user task. The formal parameters are described in *TIBCO Business Studio Process Modeling User's Guide*.

User Task ParametersUser task parameters are properties of a specific user task within a business process, and they are mapped to data fields or process parameters using the **Interface** tab of the user task's Properties view.

The Mode Property of User Task Parameters

The Mode property of a user task parameter refers to whether it is an **In**, **Out**, or **In/Out** parameter. The Mode is set differently, depending on whether the parameter is derived

from a process parameter or a data field:

- **Process Parameters** Mode value set on the parameter's Properties view.
- **Process Relevant Data** Mode value set on the user task's Interface Properties view.

Using Data Fields and Parameters

Incoming data can populate form controls with initial values or dynamically set runtime values for certain control properties, such as the text of a control label, or whether it is initially visible on the form. Outgoing data are submitted when the user clicks the **Submit** button. This data can be used in various ways in subsequent phases of the business process, sent to an external process, or written to a persistent store.

Relationship between User Task Parameters and Form Parameters

A user task and a form have their own separate models for the input and output parameters for the task or form. When a form is created from a user task, the form parameters are automatically set to match those in the user task.

The Synchronize Form operation on the user task again updates the form parameters to match those in the user task if there have been any changes since the form was originally generated. User task and form parameters must match on name, data type, multiplicity, and direction. The order is irrelevant. OUT-mode user task process parameters are represented as INOUT form parameters to support the 'Close' operation, which stores the intermediate values for later reuse prior to final form submission.

Data Types for Data Fields and Process Parameters

Data fields and process parameters must be set to one of the predefined primitive data types supported by XPDL to be mapped to a form parameter.

Primitive Data Types

The following simple data types can be used for a data field or process parameter that is mapped to a form parameter:

String A set of alphanumeric characters, with a specified maximum length

Decimal NumberA decimal number with a specified maximum length and specified number of decimal places

Integer NumberAn integer value, with a specified maximum length

BooleanTrue or false

Date TimeA combination of date/time

TimeTime only

DateDate only

Icons for Data Types

In the case of data fields, an icon appears next to each data field in the Project Explorer to identify the data type of the field.

Data Types

Icon	Data Type
	Text
	Decimal Number
	Integer Number
	Boolean
	Date Time
	Time
	Date

Data Field Levels

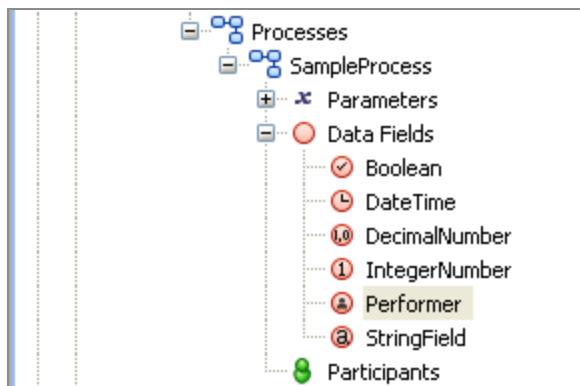
A data field can be created at either of the two different levels of the project hierarchy in the Project Explorer, the process level or the package level. The level determines the scope of the data field.

Data Fields at the Process Level

Data fields created at the process level appear under a business process in the Project Explorer. The fields are available to any form associated with a user task within that process, but not to forms in other business processes in the package.

Data fields created at this level appear beneath the business process in the Package Explorer.

Figure 17: Data Fields at the Process Level

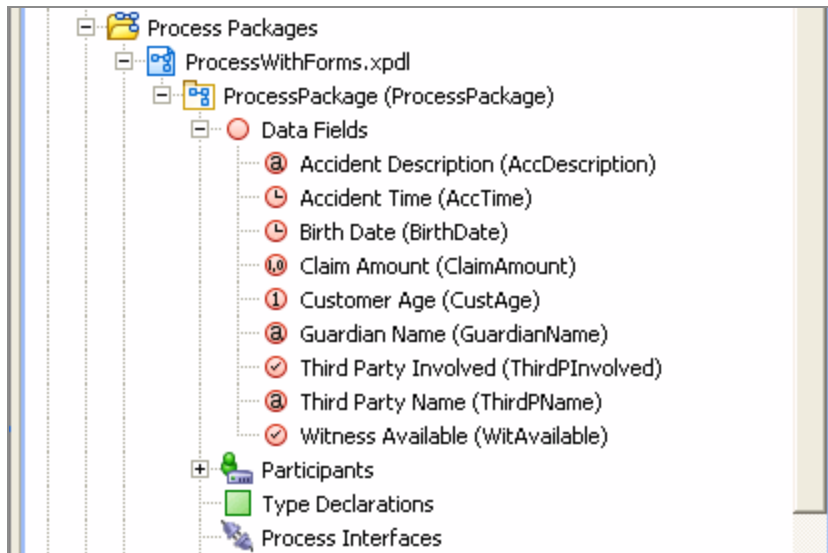


Data Fields at the Process Package Level

Data fields created at the package level appear under a process package in the Project Explorer and are available to any form in any business process within the process package. In other respects, data fields at the package level are identical to those at the process level.

Data fields created at this level appear beneath the process package in the **Package Explorer**.

Figure 18: Data Fields at the Process Package Level



Adding iProcess System Fields to a Form

Business processes in the iProcess runtime environment have implicit system data fields. These system fields can be used as form parameters and are associated with controls on the form.

To add a system field to a form, follow these steps:

1. Open the form in the Form Designer window.
2. In the Outline View for the form (viewed in the hierarchical tree mode), right-click **Data** and click **New Parameter**.
3. In the **New Parameter** dialog, enter the name of the implicit data field.



Warning

Enter the name of the implicit data field exactly as specified in the [Complex Data](#).

4. Enter the data type and length of the field. If unsure of these values, consult the iProcess documentation.
5. Click **Finish**.

6. The form parameter is now available and can be associated with a control on the form, for instance as the incoming value of label control or text field.

See [TIBCO iProcess System Fields](#) for a list of iProcess system fields supported by TIBCO Business Studio Forms.

Complex Data

The classes defined in the business object model (BOM) are represented in the runtime JavaScript API. Parameters can be defined as external types to classes defined in a BOM, either in the same project as the form or in dependent projects. Each attribute defined in a class results in the corresponding `get[AttributeName]()` and `set[AttributeName](value)` methods. For example, a BOM class **Customer** that had a **firstName** attribute generates a **Customer** class with the following methods:

```
String getFirstName();  
void setFirstName(String firstName);  
Refer to TIBCO Business Studio Forms User's Guide for API details.
```

Layout Properties for Panes

Horizontal Pane

In runtime environments that use TIBCO General Interface, a horizontal pane can contain either controls or child panes, but not a mixture of the two.



Warning

For the GI runtime, all immediate children of a horizontal pane must be either all panes or all controls. To model a layout with a mixture of controls and panes you should create an extra pane around each control.

Tabbed Pane

Although the Layout tab for panes contains the same fields as the Layout tab for controls, there is one minor difference: for **Tabbed Panes**, default values are given for width and height so that the tabbed pane renders properly in the preview tabs and at runtime. The

default values can be changed, but for forms which target the GI runtime environment, the size fields must contain a value. This applies only to the enclosing tabbed pane itself. The sub-panes can be left to auto-size themselves. That is, you can leave their width and height blank on the Layout tab of the Properties view.

The General Interface runtime environment for TIBCO Business Studio Forms requires the following elements:

- **Explicitly sized tabbed panes** It is up to you to set an appropriate width and height for the tabbed pane, one that accommodates both the tallest and the widest of the child panes. If you clear the explicit height or width setting, this results in a problem marker appearing in the Problems view when the Eclipse Builder runs, if the pane targets the GI presentation kind.
- **A child label position of top** Although the Child Labels tab allows you to set position values other than top, doing so results in a problem marker appearing in the Problems view when the Eclipse Builder runs. If this occurs, you can use the Quick Fix to restore the correct label position (select marker in **Problems View > Context Menu > Quick Fix**).

For the TIBCO General Interface runtime environment, tabbed panes are not resized automatically. You may have to click through the tabs or your tabbed panes, and look for the largest one. Make sure that the tabbed pane is big enough to accommodate the largest child pane it contains.

The Outline view clearly shows the direct children of a tabbed pane, which are always other panes.

Grid Pane

Grid panes in GI Forms do not show the original order again once you choose to sort a column.

Localizing a Form

The locale option list in the **GI Preview** is a convenience for checking the results of various locale settings. At runtime, the locale of an actual user is set on the user's system or in the user's browser.

Styling Forms Using Cascading Style Sheets

While you can control some layout and font properties via the form model **Property** tabs, it is also possible to specify additional CSS classes that are applied to form components at the form, pane, and control level. This approach provides more flexibility and opportunities for reuse of style information above manually setting properties at the Form model level.

Setting CSS Classes

The **General** property sheet for the form, panes and controls includes an input box to enter the CSS class for the given component. The value can be either a single value, or a space-separated list of CSS classes. When the component is rendered in the web page, the CSS classes entered here is added to the HTML along with other built-in CSS classes. You can update the value of the CSS class for a form, pane or control using bindings, computation actions, or using the API.

Using an External CSS Resource

Using the **Resources** property sheet for a form you can reference one or more external CSS files from the form. When added as an external reference, the CSS is loaded before the form loads. To load an external CSS file in a form:

- Place the CSS resource within the **Presentation Resources** special folder.
- Select the root of the form by either clicking in the background of the canvas or selecting the root node in the outline view.
- Select the **External Resources** property tab.
- Click the **Add (+)** button to add a reference to the CSS.

Best Practices

- Use `.TibcoForms` in class selectors.

The root node of each form specifies a class `TibcoForms`. With it, you can write CSS selectors that are specific to Forms and do not conflict with other elements on the

page. For example, suppose you have a CSS class **highlight** that you apply to a pane. The corresponding CSS rule may be written as follows:

```
.TibcoForms .highlight {background-color: yellow;}
```

This ensures that the **highlight** class only gets applied to elements within a form.

- Share CSS between forms.

You can share the same CSS between multiple forms to cut down on duplication. For this, add a reference to the shared CSS from one or more forms.

- Use **#JSX** in the selector.

GI forms are rendered into GI applications where many CSS rules are already declared using **#JSX** in the selector. That means if you want to override some of the built-in CSS rules, it is necessary to use CSS selectors such as:

```
#JSX .TibcoForms .control-textinput {font-family: Helvetica;}
```

Using a selector with a leading **#JSX** is necessary in many cases to get a higher specificity in evaluating the CSS selector.

Examples

A vertical pane might use a set of classes, such as the following:

```
pane pane-vertical
  pane-label
  pane-content
    component control-textinput required
      label
      container
        widget-text
        hint
    component control-date
      label
      container
        widget-date
        hint
```

You may use the following selectors:

.pane-vertical .hint Applies to hints within vertical panes

.control-date .label Applies to labels of Date controls

.pane-vertical .required Applies to required controls within a vertical pane

.pane-vertical .pane-horizontal .label Applies to the labels in horizontal panes nested within vertical panes.

Cascading Style Sheets - Advanced Samples

This section describes advanced tasks you can perform using TIBCO iProcess Workspace (Browser).

Import the Forms Advanced Samples

The advanced samples are available on the TIBCO Developer Network site. To download and install the advanced samples, do the following:

1. Navigate to http://developer.tibco.com/business_studio/samples.jsp.
2. Under **PRODUCT SAMPLES**, click **View all Product Samples**.
3. Click **Form Samples** to go to the Forms Samples page.
4. Under **Forms Advanced Samples**, click **Download and Install** to get the sample in your workspace.

Using CSS to Customize the Rendering of a Form Control

TIBCO Business Studio Forms supports the use of Cascading Style Sheets (CSS) for customizing how form controls are rendered. This section shows how to use CSS with Business Studio Forms to apply styling to a form control.

Task

Customize the rendering of a control using CSS.

Sample Project

To view the sample for this task, import the advanced sample projects as described in [Import the Forms Advanced Samples](#). The sample form and CSS file for the task described in this section are contained in the `forms.samples.layout` sample project.

- The form (`ControlRendering.form`) is at the following location:
`forms.samples.layout/Forms/CSS Samples/ControlRendering/`
- The CSS file (`ControlRendering.css`) is at the following location:
`forms.samples.layout/Presentation Resources/css/`

You can double-click the form and CSS filenames (as well as those of other project resources) in the Project Explorer to open them in the editor. There, you can examine their contents and use them as models for your own projects.

Explanation

This task covers the case where you want to apply special styling to a specific control in a form.

To design the rendering of a control, understand how the control is rendered in the browser. TIBCO Forms makes use of CSS classes attached to the HTML DOM nodes to control rendering. Generally, knowing which actual HTML elements are being used in the rendering is not necessary. You should try to use only the CSS classes to devise CSS selectors in your stylesheets, as this approach is the most portable across different target platforms.

The following is a representation of the CSS classes that are used to render a control, and their relationship to one another within the nested DOM:

```
-component, customclass
  -label
  -container
    -control
    -hint
```

The name of a CSS class entered in the design time model is *customclass*.

To implement this task, perform these steps:

1. Create a form that contains one or more controls.

2. Link the form to a custom CSS stylesheet. A custom CSS stylesheet called `ControlRendering.css` is provided in the `forms.samples.layout` project in the directory `Forms/Presentation Resources/css/`

**Tip**

To create a CSS file in your project In the Project Explorer, right-click the **Presentation Resources** folder for your project, and click **New > File**. The **New File** dialog opens, where you indicate the parent folder that contains the CSS file for this form, and the file name. If there is already a **css** folder within your **Presentation Resources** folder, you can choose that one or, if not, create a folder with that name. But whether you use a subfolder, and if so, what it is named is unimportant. The CSS file must be placed in or under the **Presentation Resources** folder and that its filename must end with the extension `.css`. When you click **Finish**, the CSS file is created and opened in the editor.

**Tip**

To link a form to a CSS stylesheet Be sure the CSS file is already present in the **Presentation Resources** folder. Then, in the **Properties** view for the form, click the **Resources** tab. Click the **Add (+)** button to add a resource. The **Pick Resource** dialog opens, displaying a list of the resources currently residing in the **Presentation Resources** folder, including CSS files, JavaScript, and image files, if any. Select the desired CSS file, and click **OK**. The CSS file is now added as a resource to the form. The definitions it contains are used to render the form in HTML.

The remainder of these instructions assume that your CSS file matches the sample file in the `forms.samples.layout` project.

3. With the form open and visible in the editor, click one of the controls on the form to open the **Properties** view for the control.
4. Enter **control-rendering** in the **Style Class Name(s)** box on the **General** tab of the **Properties** view for the control.
5. Change the label font properties for this control. Add the following lines in the linked CSS stylesheet:

```
#JSX .highlight .label {
    color: #FF0000;
    font-family: Helvetica, sans-serif;
```

```
font-size: 12px;
font-weight: bold;
}
```

Note the CSS selector used here is **#JSX .highlight .label**. This selector is needed when the form is rendered using GI, which is the rendering used in TIBCO iProcess Workspace.

6. Put a border around the highlighted control and change the background color. Add the following lines to the linked CSS stylesheet:

```
#JSX .highlight {
border-style:solid;
border-width: thin;
background-color: #DDFFDD;
}
```

Changing the Background Color of a Control based on its Value

This task shows how to customize the background color of a control using a computation action and CSS classes.

Task

Change the background color of a control based on its value.

Sample Project

To view the sample for this task, import the advanced sample projects as described in [Import the Forms Advanced Samples](#). The sample form and CSS file for the task described in this section are contained in the `forms.samples.controls` project.

The form (`SetBackgroundColorForControl.form`) is at the following location:

```
forms.samples.controls/Forms/setBGColor/
```

The CSS file (`custom.css`) is at the following location:

```
forms.samples.controls/Presentation Resources/css/
```

You can double-click the form and CSS filenames (as well as those of other project resources) in the Project Explorer to open them in the editor. You can examine their contents and use them as models for your own projects.

Explanation

This topic covers the case where you want to apply a background color to a given control in a form based on the value of the control.

To implement this task, you need to know:

- How to specify a custom CSS document and refer it in the form
- How to add a computation action that is targeted to a property of the control

To implement the task, perform these steps:

1. Create a form with one or more controls.
2. Add following classes to the custom CSS document and refer to the document in the form:

```
#JSX .normalbg {
    background-color: #808080;
}
#JSX .warningbg {
    background-color: #00FF00;
}
#JSX .problembg {
    background-color: #FF0000;
}
```

3. Add a computation action for the **Style Class Name(s)** property in the **General** Properties view for the form.

Provide the following JavaScript code for this action, and select the **update** event of this control.

```
var value = parseInt(control.textinput1.getValue());
var bgclass = "normalbg";
if ( value <= 100 ) {
    "normalbg";
} else if ( value > 100 && value <= 500 ) {
    "warningbg";
}
```

```

    } else if ( value > 500 && value <= 1000 ) {
        "problembg";
    }

```

4. Preview the form.

Provide an integer value between 0 - 100 and the background color for the control is set to gray.

Provide an integer value between 101 - 500 and the background color for the control is set to green.

Provide an integer value between 501 - 1000 and the background color for the control is set to red.

TIBCO iProcess System Fields

TIBCO Business Studio Forms supports the TIBCO iProcess system fields listed in this section. These fields can be added as form parameters by following the procedure described in [Adding iProcess System Fields to a Form](#). The following list shows the name of the system field, whether the field is read-only or read-write, the data type, and length. Read-only fields should be used as **In** parameters only. Read-write fields can be used as **In** or **In/Out** parameters.



Tip

For more information on TIBCO iProcess system fields, see the TIBCO iProcess documentation.

TIBCO iProcess System Fields Supported as Form Parameters

System Field Name	Read only or read/write	Data type	Length
SW_CASEDESC	read/write	String	24
SW_CASENUM	read only	Integer Number	15
SW_CASEREF	read only	String	20

System Field Name	Read only or read/write	Data type	Length
SW_CP_INCPERIOD	read/write	Integer Number	4
SW_CP_INCREMENT	read/write	Integer Number	4
SW_CP_NUMINC	read/write	Integer Number	3
SW_CP_PERIODTYP	read/write	String	1
SW_CP_VALUE	read/write	Integer Number	3
SW_DATE	read only	Date	n/a
SW_GEN_IDX	read/write	Integer Number	6
SW_HOSTNAME	read only	String	24
SW_IP_INCPERIOD	read/write	Integer Number	4
SW_IP_INCREMENT	read/write	Integer Number	4
SW_IP_NUMINC	read/write	Integer Number	3
SW_IP_PERIODTYP	read/write	Integer Number	1
SW_IP_VALUE	read/write	Integer Number	3
SW_PRODESC	read only	String	24
SW_PRONAME	read only	String	8
SW_QRETRYCOUNT	read only	Integer Number	15
SW_STEPDESC	read only	String	24
SW_STEPNAME	read only	String	8

System Field Name	Read only or read/write	Data type	Length
SW_TIME	read only	Time	n/a
SW_MAINCASE	read only	Integer Number	10
SW_MAINHOST	read only	String	24
SW_MAINPROC	read only	String	8
SW_PARENTCASE	read only	Integer Number	10
SW_PARENTHOST	read only	String	24
SW_PARENTREF	read only	String	64
SW_PARENTPROC	read only	String	8
SW_QPARAM1	read/write	Variable	n/a
SW_QPARAM2	read/write	Variable	n/a
SW_QPARAM3	read/write	Variable	n/a
SW_QPARAM4	read/write	Variable	n/a

Validation Errors

This section describes the rules applied by the Validation Builder.

General Interface Errors

These are rules that apply specifically to form models targeted at the TIBCO General Interface runtime.

Rule	Message	Quick Fix?	Description
Explicit label properties	GI form elements may not supply explicit label properties other than 'visible'	No	TIBCO General Interface can only support label visibility for panes and controls. All other label-related settings must be inherited from the containing form or Child Label of the pane.
Horizontal pane child homogeneity	GI horizontal panes cannot mix panes and controls	No	TIBCO General Interface horizontal panes must contain only sub-panes or controls, not a mixture of both.
Tabbed pane explicit size	GI tabbed panes must have an explicit width height	No	In TIBCO General Interface, a tabbed pane with no explicit size is not visible.
Tabbed pane label position	GI tabbed panes must have a child label position of 'top'	Yes	TIBCO General Interface tabbed panes only support a child label position of 'top'. Other positions are ignored.

Using the GI Preview Mode on Linux

On the Linux platform, the GI Preview mode for forms in the Form Designer only works if the Mozilla Gecko rendering engine version 1.8.1.3 or later is installed on the machine. GI Preview does not work on Linux, for instance, with Mozilla Gecko version 1.7.3.

Upgrading to the Current Version of Mozilla Gecko on Linux

To upgrade an older version of Mozilla Gecko to the supported version, perform these steps:

1. Download `xulrunner-1.8.1.3.en-US.linux-i686.tar.gz` from the Mozilla Web site (<http://releases.mozilla.org/pub/mozilla.org/xulrunner/releases/1.8.1.3/contrib/linux-i686>)
2. Unpack the tarball to a new directory using the following command:

```
tar -xzf xulrunner-1.8.1.3.en-US.linux-i686.tar.gz
```
3. Navigate to the install directory and register XULRunner with the system by running the following command:

```
xulrunner -register-global
```

 (to install for all users; must be run as root)
or

```
xulrunner -register-user
```

 (to install for one user only)
4. Start TIBCO Business Studio and confirm that you can preview your form.

**Tip**

More details about XULRunner can be found at http://developer.mozilla.org/en/docs/XULRunner_1.8.0.4_Release_Notes.

Deploying Process Packages, Forms, and Dependent Resources

You can deploy either the whole process package, or any of the forms that are in the package. You can also choose between the drag-and-drop deployment or a menu-driven deployment mechanism. In both cases, dependent resources are deployed when a form is deployed.

Deploy a Process Package to the iProcess Server

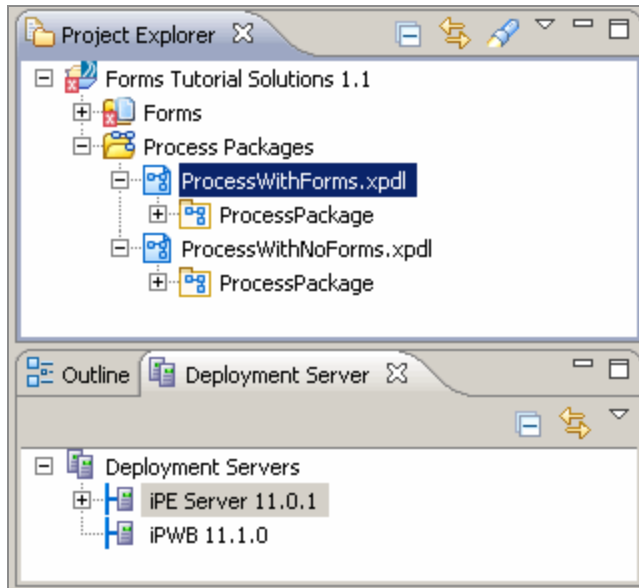
To deploy the whole process package to an iProcess server, do the following:

1. Make sure that the server is configured. Forms associated with the process also get deployed if the deployment server is configured for the iProcess Workspace server configuration details.

See the *TIBCO Business Studio* documentation to configure the server.

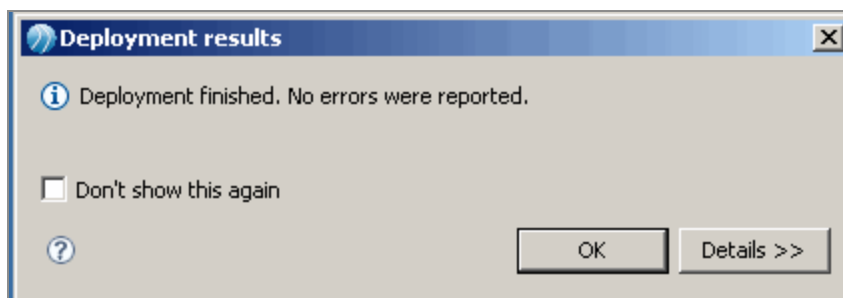
2. Select the process package icon in the Project Explorer view, then drag and drop it on the iProcess server icon in the Deployment Server view.

Figure 19: Drop the Process Package Icon on the iProcess Server to Deploy



The Deployment results dialog opens, showing the deployment status.

Figure 20: Deployment Results



3. Click **Details >>** to see the detailed deployment result.

Deploy a Form to the iProcess Workspace Server

If you are working in a development mode and need to deploy a form to an iProcess Workspace server, do the following:

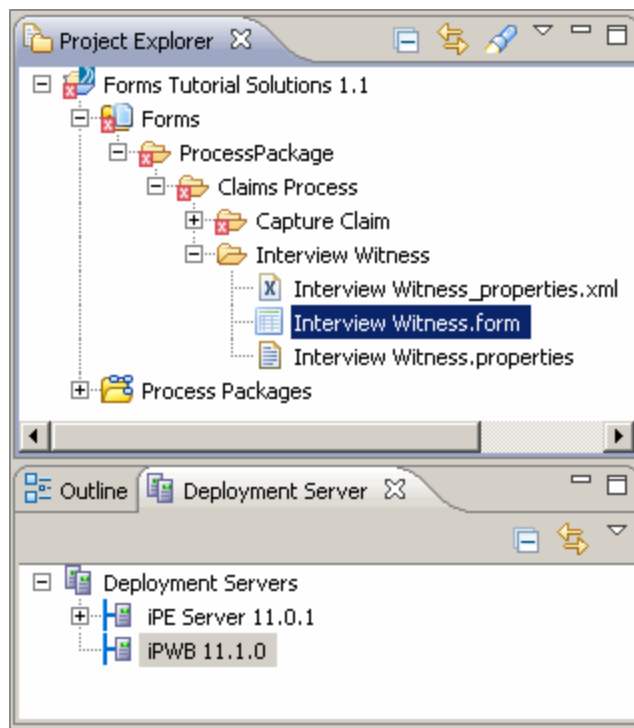
1. Make sure that the server is configured.

See *TIBCO Business Studio iProcess Developer's Guide* to configure the server.

2. Connect to the iProcess Workspace Server. Click the **Connect** mouse button one option (MB1) available for the iProcess Workspace Server icon in the Deployment Server view.
3. Drag the form icon in the Project Explorer view, and drop it on the iProcess Workspace Server icon in the Deployment Server view.

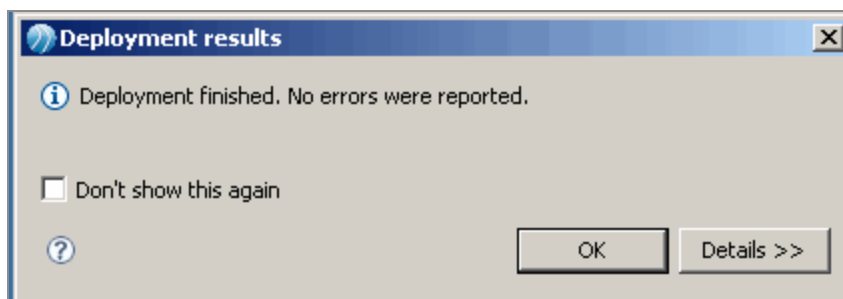
All the dependent artifacts of the form such as the GI Resource bundles, images, and JavaScript files are automatically deployed along with the selected form.

Figure 21: Drop the Form Icon on the iProcess Workspace Server to Deploy



The Deployment results dialog opens, showing the deployment status.

Figure 22: Deployment Results



4. Click **Details >>** to see the detailed deployment result.

Deploy Using the Deployment Wizard

When the user selects the option **Deploy Module** from the deployment server, the deployment wizard displays only projects, folders, and *.form resources, while the other resource types are filtered out by the selection wizard. When selecting a form to deploy, all dependent resources are deployed as well, including :

- GI resource files generated from the properties files associated with the .form resource
- Image files referenced by relative URLs in the image controls in the form model
- Additional image files referenced from the form resources Properties tab
- JavaScript files referenced by relative URLs in the form model

The user picks the *.form resource from within a Forms special folder in the project. The actual resources deployed come from the special folder of the .forms deployment module. This is a hidden folder, and the folder structure mirrors that of the visible forms special folder.

To deploy the forms and their artifacts using the deployment wizard:

1. Click the **Deploy Module** mouse button one (MB1) menu option available for the deployment server to open the deployment wizard.
2. Select the forms and its artifacts to deploy with the wizard.
3. Follow the wizard options to complete the deployment.
4. The deployment results dialog opens, showing the deployment status.

When a process package is deployed using the iProcess deployment server, the list of dependent form resources is generated and is deployed as well. The form resources are listed in the final screen of the process deployment wizard.

Supported Forms Formats

TIBCO Business Studio 3.1 can migrate forms created in the earlier versions of Business Studio. The earlier versions of TIBCO Business Studio do not support forms created in TIBCO Business Studio 3.1.

The iProcess Workspace (Browser) 11.1 supports forms created in all the versions of Business Studio. The earlier versions of iProcess Workspace (Browser) do not support forms created in TIBCO Business Studio 3.1.

Supported Form Formats

Design Time		Runtime		
Business Studio	Forms Format	iProcess Workspace (Browser)	Forms Format	iProcess Engine
2.2 (Feb'08)	1.0	10.7	1.0	10.6, 11.0
3.0 (May'08)	1.0	11.0, 11.1	1.0	11.0
3.1 (Dec'08)	2.0	11.1	2.0, 1.0	11.0
3.2.0 (July'09)	2.0	11.2	2.0, 1.0	
3.4.2 (Sept'10)	2.0	11.3	2.0, 1.0	
3.5.2 (Oct'11)	2.0	11.4.	2.0, 1.0	

Unsupported Forms Features

iProcess WorkSpace Browser does not support the following features:

- iProcess does not support complex parameters, so form parameters can only be of primitive type.
- Only GI forms are supported for iProcess WorkSpace Browser .
- The formsclient API is not supported for iProcess WorkSpace Browser.
- Advanced pane rendering properties, such as Static Rendering, Defer Rendering, Defer Initialization, Text Only are not supported.
- Rendering of forms on mobile devices is not supported.
- Properties, such as "Always render controls", "Default Sort Order," and "Default sort column" are not supported in the grid pane.
- Custom controls are not supported.

Managing Monitor Event Request (MER) Messages

This section describes how to create or edit Monitor Event Request (MER) messages, which are used to specify events published to JMS.

Introduction

The TIBCO iProcess Engine can be configured to publish audit events to JMS for specific steps of selected procedures, providing associated field data values. External applications can then consume the audit events that are published to JMS.

The specification of the published events is defined in a Monitor Event Request (MER) message, which is an XML document whose schema is defined in `SWMonitorList.xsd`.

Structure of MER XML

The following is an overview of the elements in the schema structure defined in `SWMonitorList.xsd`:

<FullImport>	Specifies whether to overwrite or append to existing MER definitions.
<MonitorDetail>	A <MonitorDetail> element exists for each procedure to be monitored.
<ProcedureName>	The procedure to be monitored.
<NodeName>	The node on which the procedure resides.
<GlobalFieldList>	A list of field names whose data will be exported for every event.
<MonitorList>	A list of <MonitorLine> elements specifying activities, steps, and fields to monitor.

<MonitorLine>	A grouping of events, steps, and fields to monitor.
<ActivityList>	List of audit events to be published (Case Start, Release, etc.); "-1" means all events.
<StepList>	List of steps for which the events specified in <ActivityList> will be published; "\$ALL\$" means all steps.
<FieldList>	List of fields whose data will be published for the event/step combination (there is no wildcard for "all" fields due to negative impact on system performance).

An example MER message is:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><ProcedureMonitor
xmlns="http://bpm.tibco.com/2004/IAP/MER/1.1/MER"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://bpm.tibco.com/2004/IAP/MER/1.1/MER
SWMonitorList.xsd">
  <SchemaVersion>1.1</SchemaVersion>
  <MessageType>MER</MessageType>
  <FullImport>true</FullImport>
  <MonitorDetail>
    <Procedure Name="CARPOOL">
      <NodeName>v11</NodeName>
    </Procedure>
    <GlobalFieldList>
      <Field Name="ALLOC"></Field>
    </GlobalFieldList>
    <MonitorList>
      <MonitorLine>
        <ActivityList>
          <Activity Num="0"></Activity>
        </ActivityList>
        <StepList>
          <Step Name="AUTHORIZ"></Step>
        </StepList>
        <FieldList>
          <Field Name="ENDTIME"></Field>
        </FieldList>
      </MonitorLine>
    </MonitorList>
  </MonitorDetail>
</ProcedureMonitor>
```

Manage Monitor Event Request Tool

The iProcess Workspace (Browser) application provides a tool that can be used to create new, or edit existing, MER messages on the iProcess Engine. To access this tool, from the procedure list:

- select **Tools > Manage Monitor Event Request**, or
- click the  icon on the toolbar.

These selections cause the **Manage Monitor Event Request** dialog to be displayed, which provides options to do the following:

- [Create New Monitor Event Request](#) - Use this option if you want to create a new MER message. This sets the <FullImport> element in the MER XML to true, which causes the new MER message to overwrite any existing MER message on the iProcess Engine. (To view the existing MER message on the iProcess Engine, you can select the **Edit Monitor Event Request on the Server** option, then cancel out of it, if desired.)
- [Edit Monitor Event Request on the Server](#) - Use this option if you want to edit an existing MER message on the iProcess Engine. This allows you to add additional MonitorDetails to any existing MonitorDetails, or to edit an existing MonitorDetail. This sets the <FullImport> element in the MER XML to true.
- [Import Monitor Event Request XML](#) - Use this option to either import or construct the MER XML manually. When using this option, you can set the <FullImport> element to true to overwrite an existing MER, or to false to append to an existing MER.

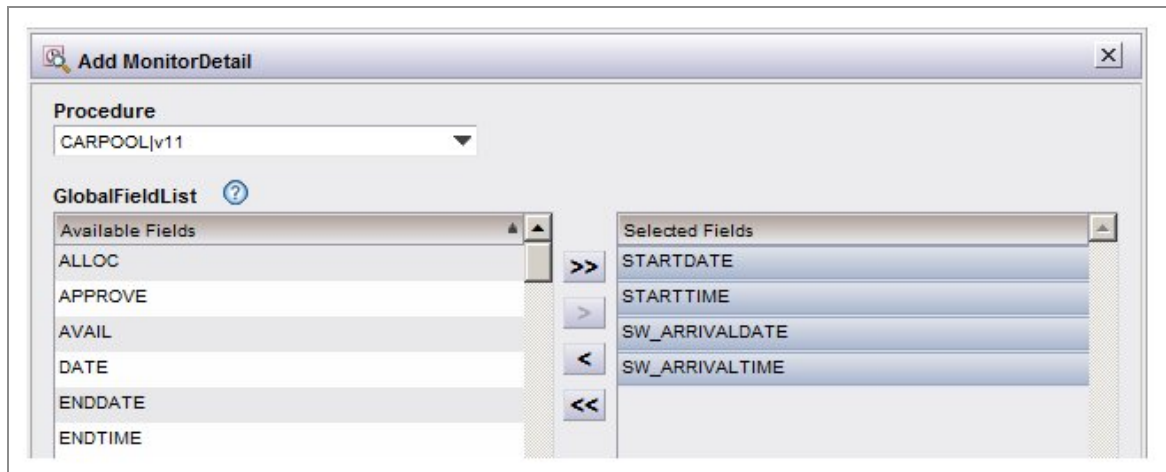
Create New Monitor Event Request

To create a new MER message, and overwrite any existing MER message on the iProcess Engine:

1. From the **Manage Monitor Event Request** dialog, select the **Create New Monitor Event Request** option.
2. On the **Monitor Event Request Definition** dialog, click the  icon.
3. From the **Procedure** field drop-down list on the **Add Monitor Detail** dialog, select the procedure whose events you want to monitor.

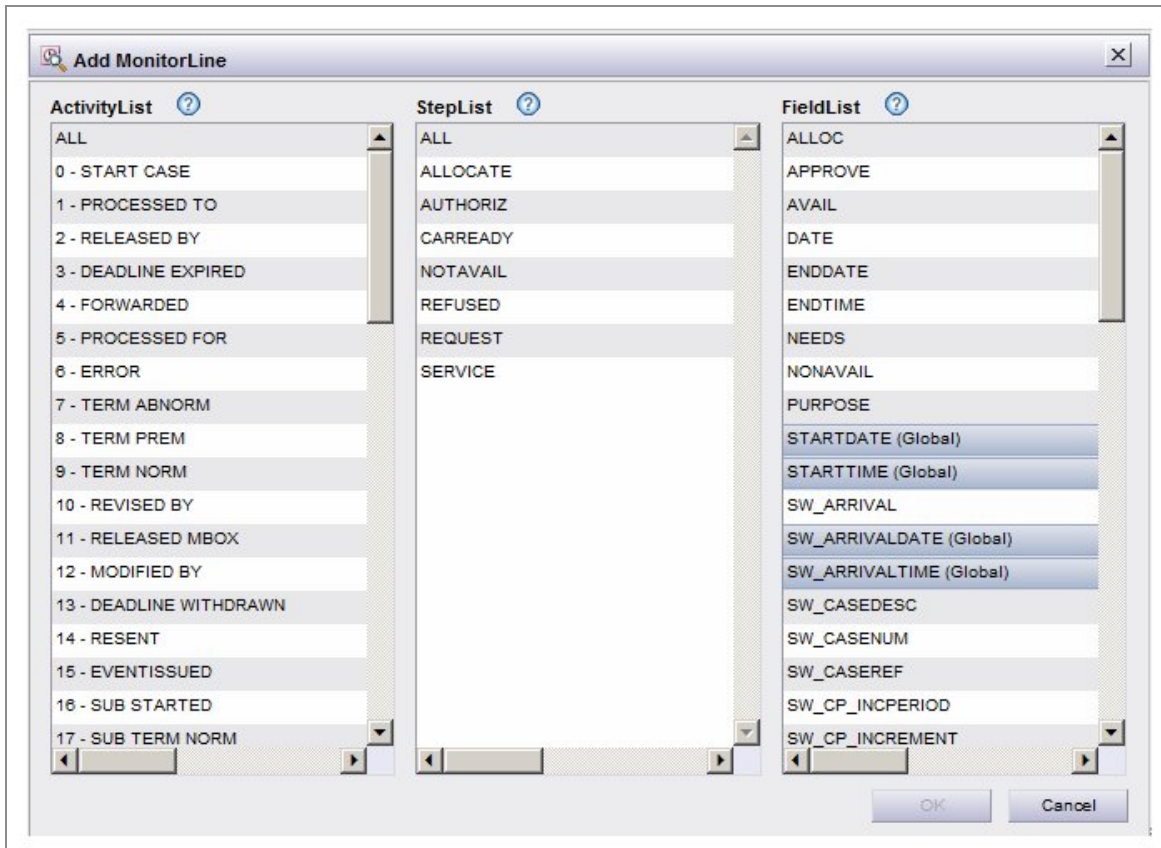
This causes the **GlobalFieldList** to be populated with all of the fields defined in the selected procedure.

4. In the **Available Fields** list, select the “global fields” whose data will be exported for every monitored event defined for this MonitorDetail (the <GlobalFieldList> element of the MER). Global fields are optional. (You can later select fields for a specific MonitorLine included in the MonitorDetail.)
5. If you have selected any global fields, move them to the **Selected Fields** list:



6. In the **MonitorList** section, click the  icon.

This causes the **Add MonitorLine** dialog to be displayed, which is used to define the <MonitorLine> element in the MER XML. This dialog displays a list of all activities (audit events), steps, and fields that are defined in the previously selected procedure:



7. From the **ActivityList**, select all of the audit events that you want to publish.

This list contains all available events on the iProcess Engine, including custom events that have been configured. You may select multiple events, or select “ALL” to publish all events. At least one event must be selected.

8. From the **StepList**, select the steps for which the selected activities (events) are to be published.

This list contains all steps defined for the procedure. You may select multiple steps, or select “ALL” to publish activities on all steps. At least one step must be selected.

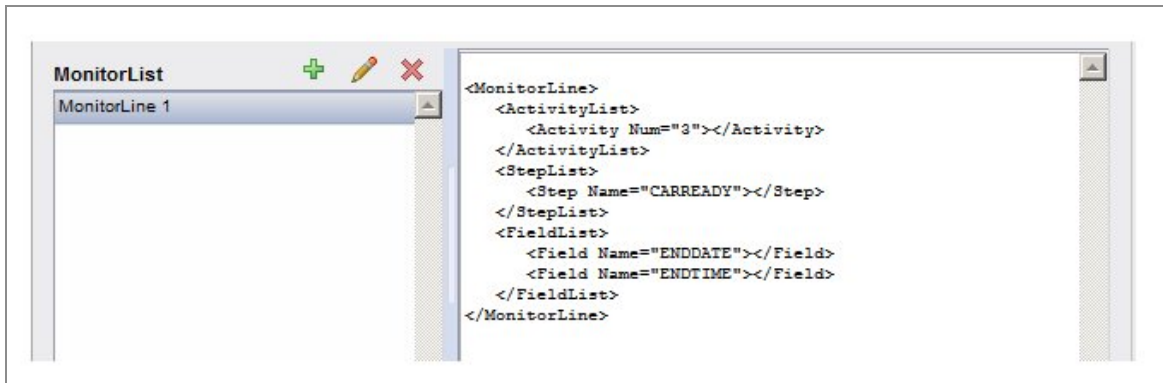
9. From the **FieldList**, optionally select fields whose data you want published.

This list contains all fields defined for the procedure. Global fields, which were selected on the **MonitorDetail** dialog (see [In the Available Fields list, select the “global fields” whose data will be exported for every monitored event defined for this MonitorDetail \(the <GlobalFieldList> element of the MER\).](#) Global fields are optional. (You can later select fields for a specific MonitorLine included in the [MonitorDetail.](#)), are automatically selected and labeled with “(Global)”. Global fields may not be deselected because they apply to all MonitorLines that are

defined. Additional fields may be selected. Data for all selected fields is published for the selected activities and steps. Selection of fields is optional.

10. After making the desired selections from the lists on the **Add MonitorLine** dialog, click **OK**.

The <MonitorLine> created by the selections on the **Add MonitorLine** dialog is displayed on the bottom of the **Add MonitorDetail** dialog:



Note that the <MonitorLine> does not include any global fields, if specified, because they apply to all MonitorLines for the MonitorDetail.

Also, because the schema does not provide the ability to specify a name for uniquely identifying each <MonitorLine> element, a name is generated for display purposes in the **MonitorList**. The generated name consists of "MonitorLine #". This name is for display purposes only and is not reflected in the XML. The MonitorLine number only signifies the position of the item in the list, if an item is removed, all the MonitorLines are renumbered — the names in the list are not static.

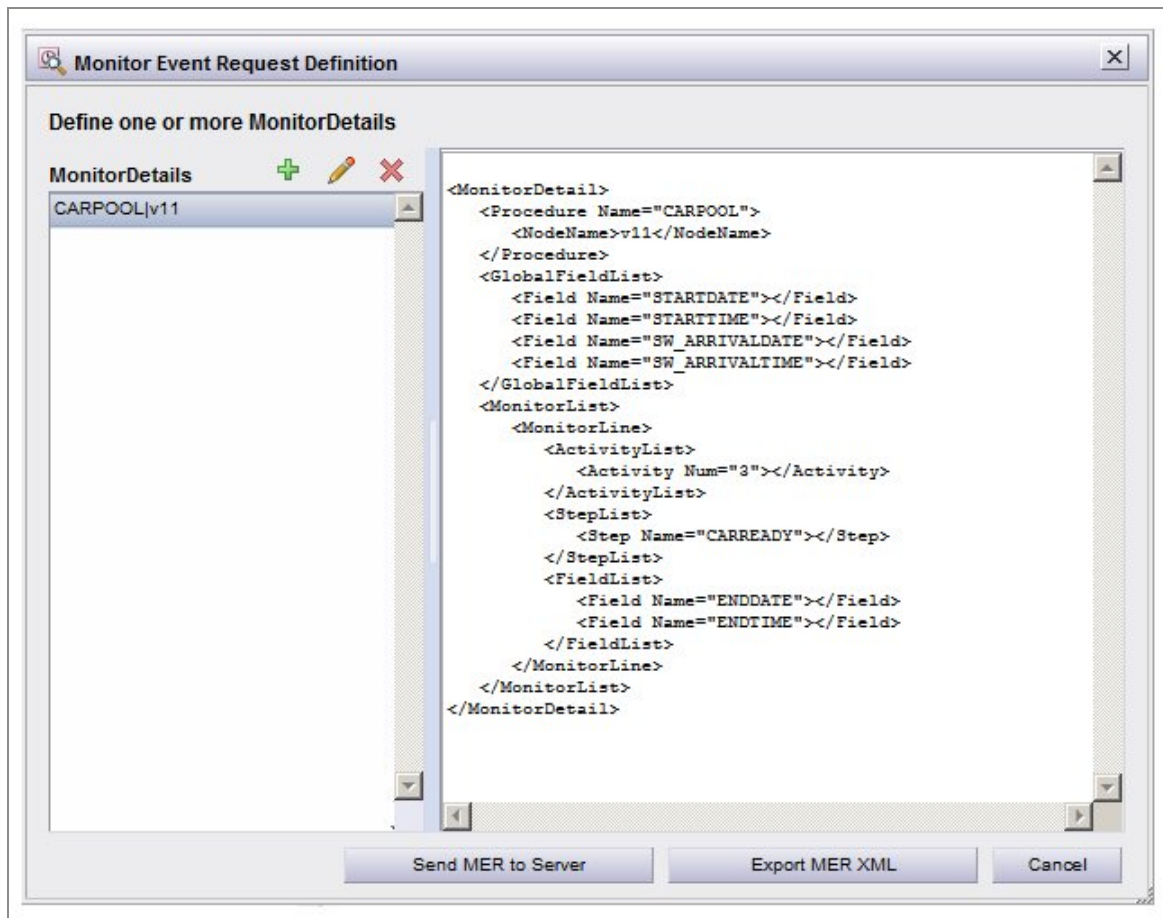


Note

You can re-visit a MonitorLine definition, if desired, by selecting the MonitorLine, then clicking the  icon. You can also delete an existing MonitorLine by selecting it, then clicking the  icon.

11. Optionally repeat [In the MonitorList section, click the icon.](#) through [After making the desired selections from the lists on the Add MonitorLine dialog, click OK.](#) if you want to define additional MonitorLines for the MonitorDetail, which could include different combinations of activities, steps, and fields.
12. On the **Add MonitorDetail** dialog, click **OK**.

The entire <MonitorDetail> that you have defined is shown on the **Monitor Event Request Definition** dialog:



13. Choose one of the following options for the new MER XML:

- **Send MER to Server** - This option writes the MER XML to the iProcess Engine, overwriting any existing MER.
- **Export MER XML** - This option causes the **Export Monitor Event Request XML** dialog to be displayed, which allow you to copy the definition of the MER XML to the clipboard for the purpose of pasting into an external file.

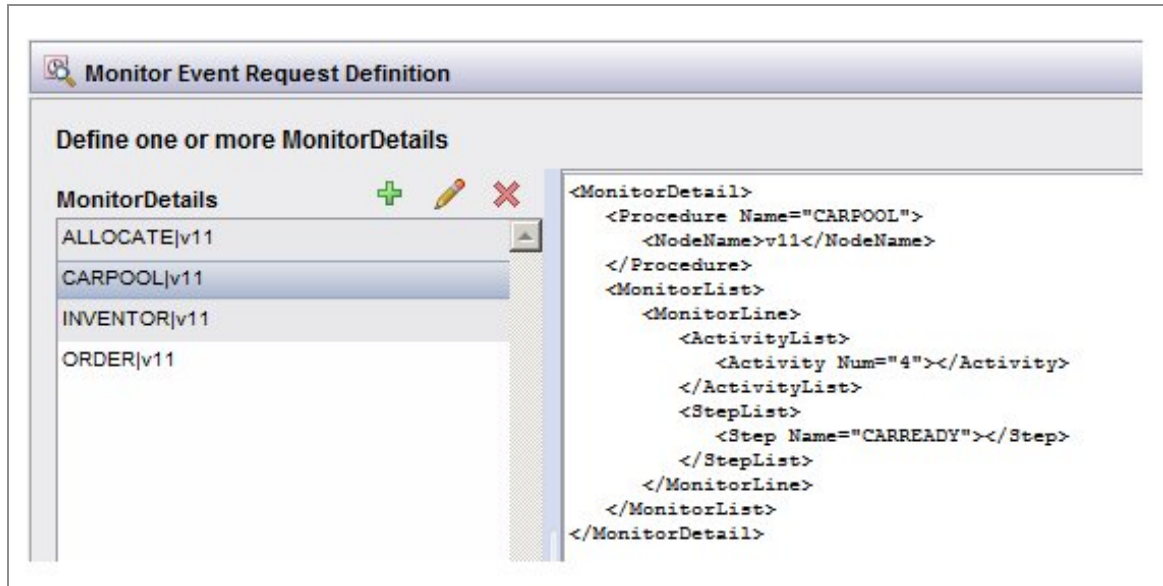
If you are running the application in Microsoft Internet Explorer, a **Copy** button is provided on the **Export Monitor Event Request XML** dialog to copy the XML to the clipboard. If you are running the application in Mozilla Firefox, there is no **Copy** button because security features prevent browser access to the clipboard.

Edit Monitor Event Request on the Server

To edit an existing MER message on the iProcess Engine:

1. From the **Manage Monitor Event Request** dialog, select the **Edit Monitor Event Request on server** option.

The **MonitorDetails** list is populated with all MonitorDetails that are currently defined on the iProcess Engine:



2. On the **MonitorDetails** list, select one of the following options:

-  - This causes the **Add Monitor Detail** dialog to display, which allows you to add a new MonitorDetail to the existing MER.
If you are adding a new MonitorDetail, follow the procedure in [Adding a New MonitorDetail to an Existing MER](#).
-  - This causes the **Edit Monitor Detail** dialog to display, which allows you to edit the MonitorDetail that is currently selected in the **MonitorDetails** list.
If you are editing an existing MonitorDetail, follow the procedure in [Editing an Existing MonitorDetail in the MER](#).
-  - This deletes the MonitorDetail that is currently selected in the **MonitorDetails** list.

After deleting an existing MonitorDetail, you can click on one of the other icons to either add a different MonitorDetail or edit an existing MonitorDetail, or you can click **Send MER to Server** to remove the deleted MonitorDetail from the MER on the iProcess Engine.

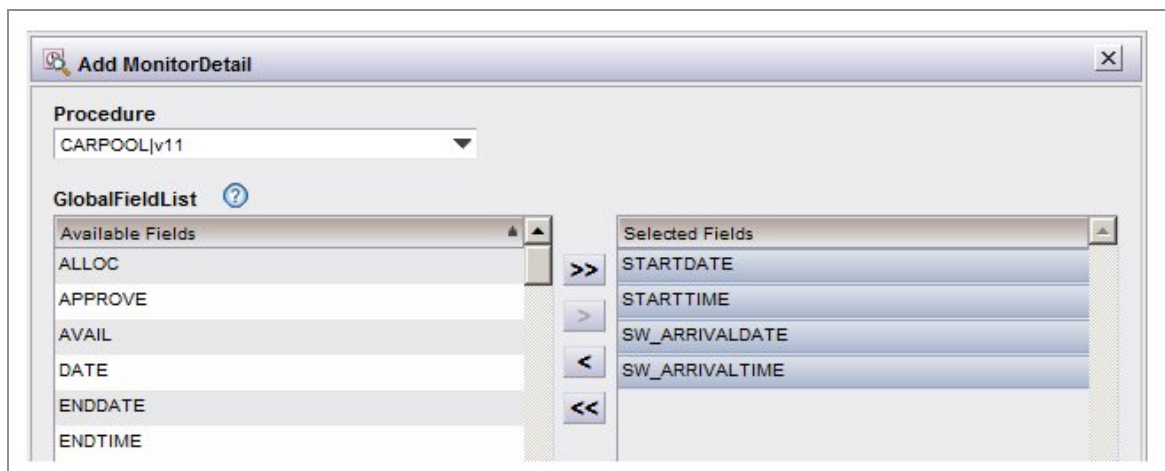
Adding a New MonitorDetail to an Existing MER

1. Click the  icon on the **Monitor Event Request Definition** dialog.
2. From the **Procedure** field drop-down list on the **Add Monitor Detail** dialog, select a procedure whose events you want to monitor.

Note that the list of procedures does not include any procedures for which there is already a MonitorDetail defined in the MER on the iProcess Engine.

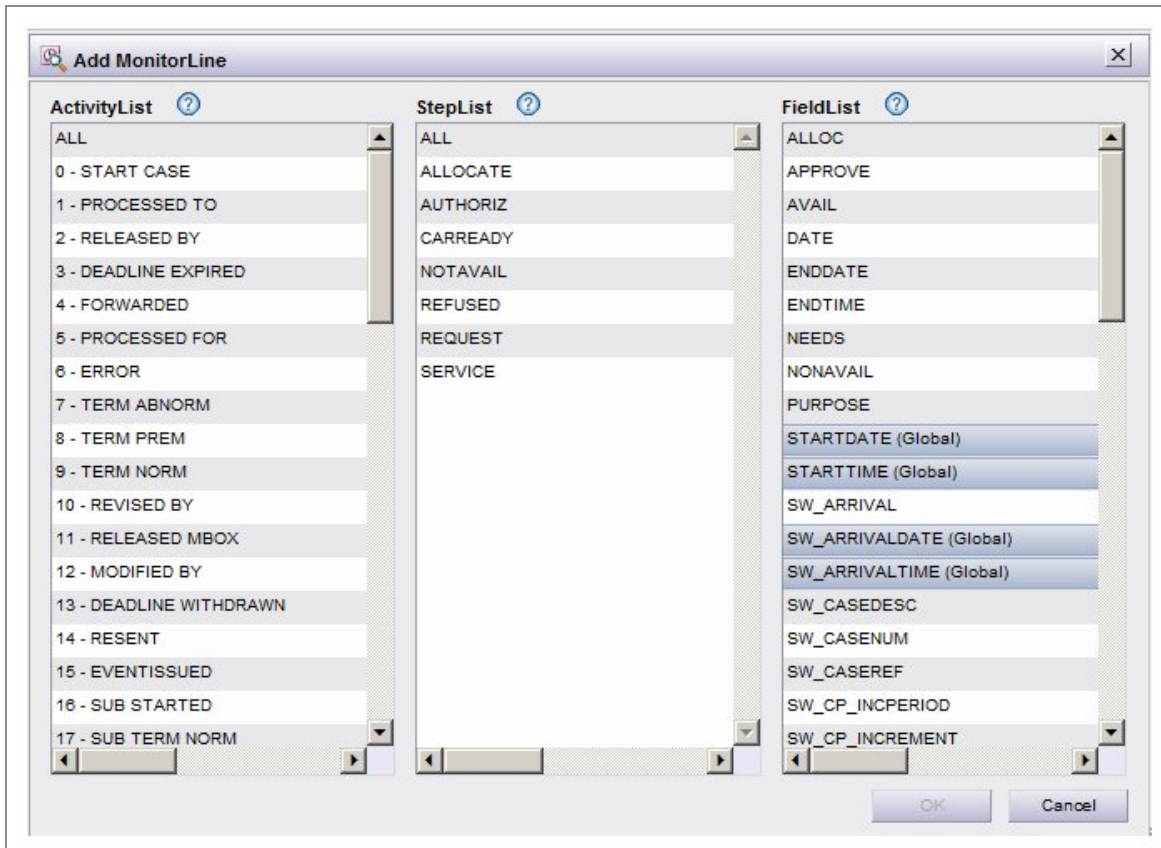
This causes the **GlobalFieldList** to be populated with all of the fields defined in the selected procedure.

3. In the **Available Fields** list, select the “global fields” whose data will be exported for every monitored event defined for this MonitorDetail (the <GlobalFieldList> element of the MER). Global fields are optional. (You can later select fields for a specific MonitorLine included in the MonitorDetail.)
4. If you have selected any global fields, move them to the **Selected Fields** list:



5. In the **MonitorList** section, click the  icon.

This causes the **Add MonitorLine** dialog to be displayed, which is used to define the <MonitorLine> element in the MER XML. This dialog displays a list of all activities (audit events), steps, and fields that are defined in the previously selected procedure:



6. From the **ActivityList**, select all of the audit events that you want to publish.

This list contains all available events on the iProcess Engine, including custom events that have been configured. You may select multiple events, or select “ALL” to publish all events. At least one event must be selected.

7. From the **StepList**, select the steps for which the selected activities (events) are to be published.

This list contains all steps defined for the procedure. You may select multiple steps, or select “ALL” to publish activities on all steps. At least one step must be selected.

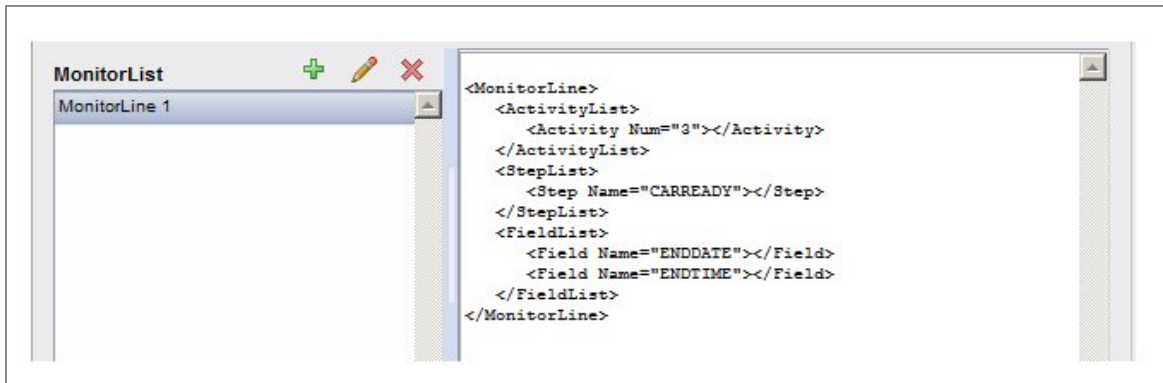
8. From the **FieldList**, optionally select fields whose data you want published.

This list contains all fields defined for the procedure. Global fields, which were selected on the **MonitorDetail** dialog (see [In the Available Fields list, select the “global fields” whose data will be exported for every monitored event defined for this MonitorDetail \(the <GlobalFieldList> element of the MER\).](#) Global fields are optional. (You can later select fields for a specific MonitorLine included in the [MonitorDetail.](#)), are automatically selected and labeled with “(Global)”. Global fields may not be deselected because they apply to all MonitorLines that are

defined. Additional fields may be selected. Data for all selected fields is published for the selected activities and steps. Selection of fields is optional.

9. After making the desired selections from the lists on the **Add MonitorLine** dialog, click **OK**.

The <MonitorLine> created by the selections on the **Add MonitorLine** dialog is displayed on the bottom of the **Add MonitorDetail** dialog:



Note that the <MonitorLine> does not include any global fields, if specified, because they apply to all MonitorLines for the MonitorDetail.

Also, because the schema does not provide the ability to specify a name for uniquely identifying each <MonitorLine> element, a name is generated for display purposes in the **MonitorList**. The generated name consists of "MonitorLine #". This name is for display purposes only and is not reflected in the XML. The MonitorLine number only signifies the position of the item in the list, if an item is removed, all the MonitorLines are renumbered — the names in the list are not static.



Note

You can re-visit a MonitorLine definition, if desired, by selecting the MonitorLine, then clicking the  icon. You can also delete an existing MonitorLine by selecting it, then clicking the  icon.

10. Optionally repeat [In the MonitorList section, click the icon.](#) through [After making the desired selections from the lists on the Add MonitorLine dialog, click OK.](#) if you want to define additional MonitorLines for the MonitorDetail, which could include different combinations of activities, steps, and fields.
11. On the **Add MonitorDetail** dialog, click **OK**.
12. On the **Monitor Event Request Definition** dialog, choose one of the following options for the edited MER XML:

- **Send MER to Server** - This option saves the edited MER message on the iProcess Engine.
- **Export MER XML** - This option causes the **Export Monitor Event Request XML** dialog to be displayed, which allows you to copy the definition of the MER XML to the clipboard for the purpose of pasting into an external file.

If you are running the application in Microsoft Internet Explorer, a **Copy** button is provided on the **Export Monitor Event Request XML** dialog to copy the XML to the clipboard. If you are running the application in Mozilla Firefox, there is no **Copy** button because security features prevent browser access to the clipboard.

Editing an Existing MonitorDetail in the MER

1. On the **Monitor Event Request Definition** dialog,, select the MonitorDetail you want to edit, then click the  icon
2. On the **Edit MonitorDetail** dialog, optionally add or remove “global fields” from the MonitorDetail. Global fields are fields whose data will be exported for every monitored event defined for this MonitorDetail (the <GlobalFieldList> element of the MER). Global fields are optional. (You can later select fields for a specific MonitorLine included in the MonitorDetail.)
3. If you only wanted to edit the global fields for this MonitorDetail, click **OK**, then proceed to [On the Add / Edit MonitorDetail dialog, click OK.](#)
4. At this point, you can either:
 - Add an additional MonitorLine to the MonitorDetail - click the  icon.
 - Edit an existing MonitorLine defined for the MonitorDetail - select the desired MonitorLine, then click the .

If you are adding an additional MonitorLine, the **Add MonitorLine** dialog is displayed; if you are editing an existing MonitorLine, the **Edit MonitorLine** dialog is displayed.

5. Define the MonitorLine on the **Add / Edit MonitorLine** dialog as follows:
From the **ActivityList**, select all of the audit events that you want to publish for this MonitorLine.

This list contains all available events on the iProcess Engine, including custom events that have been configured. You may select multiple events, or select “ALL” to publish all events. At least one event must be selected.

From the **StepList**, select the steps for which the selected activities (events) are to be published.

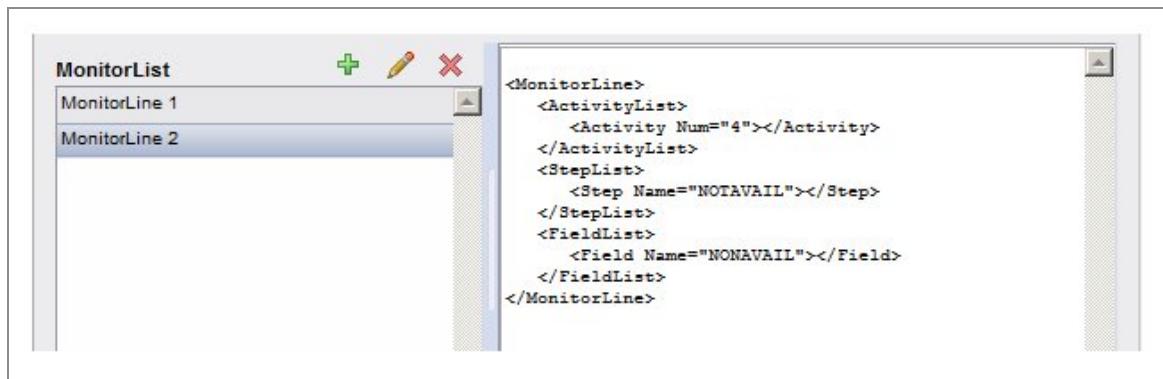
This list contains all steps defined for the procedure. You may select multiple steps, or select “ALL” to publish activities on all steps. At least one step must be selected.

From the **FieldList**, optionally select fields whose data you want published.

This list contains all fields defined for the procedure. Global fields, which were selected on the **MonitorDetail** dialog (see [On the Edit MonitorDetail dialog, optionally add or remove “global fields” from the MonitorDetail](#). Global fields are fields whose data will be exported for every monitored event defined for this **MonitorDetail** (the <GlobalFieldList> element of the MER). Global fields are optional. (You can later select fields for a specific **MonitorLine** included in the **MonitorDetail**.)), are automatically selected and labeled with “(Global)”. Global fields may not be deselected because they apply to all **MonitorLines** that are defined. Additional fields may be selected. Data for all selected fields is published for the selected activities and steps. Selection of fields is optional.

- After making the desired selections from the lists on the **Add / Edit MonitorLine** dialog, click **OK**.

All of the currently defined **MonitorLines** are shown in the **MonitorList**:



Note that the <MonitorLine> does not include any global fields, if specified, because they apply to all **MonitorLines** for the **MonitorDetail**.

Also, because the schema does not provide the ability to specify a name for uniquely identifying each <MonitorLine> element, a name is generated for display purposes in the **MonitorList**. The generated name consists of "MonitorLine #". This name is for display purposes only and is not reflected in the XML. The **MonitorLine**

number only signifies the position of the item in the list, if an item is removed, all the MonitorLines are renumbered — the names in the list are not static.

**Note**

You can re-visit a MonitorLine definition, if desired, by selecting the MonitorLine, then clicking the  icon. You can also delete an existing MonitorLine by selecting it, then clicking the  icon.

7. Optionally repeat [At this point, you can either:](#) through [After making the desired selections from the lists on the Add / Edit MonitorLine dialog, click OK.](#) if you want to define additional MonitorLines for the MonitorDetail, which could include different combinations of activities, steps, and fields.
8. On the **Add / Edit MonitorDetail** dialog, click **OK**.
9. On the **Monitor Event Request Definition** dialog, choose one of the following options for the edited MER XML:

- **Send MER to Server** - This option saves the edited MER message on the iProcess Engine.
- **Export MER XML** - This option causes the **Export Monitor Event Request XML** dialog to be displayed, which allows you to copy the definition of the MER XML to the clipboard for the purpose of pasting into an external file.

If you are running the application in Microsoft Internet Explorer, a **Copy** button is provided on the **Export Monitor Event Request XML** dialog to copy the XML to the clipboard. If you are running the application in Mozilla Firefox, there is no **Copy** button because security features prevent browser access to the clipboard.

Import Monitor Event Request XML

The **Import Monitor Event Request XML** option provides a means to either import or construct the MER XML manually.

1. From the **Manage Monitor Event Request** dialog, select the **Import Monitor Event Request XML** option.
2. On the **Import Monitor Event Request XML** dialog, either type or paste the desired XML into the dialog, then click **OK**.

The **Monitor Event Request Definition** dialog is displayed, containing the MER definition that was in the **Import Monitor Event Request XML** dialog when you clicked **OK**.

3. On the **Monitor Event Request Definition** dialog, choose one of the following options for the imported MER XML:

- **Send MER to Server** - This option saves the imported MER message on the iProcess Engine.
- **Export MER XML** - This option causes the **Export Monitor Event Request XML** dialog to be displayed, which allows you to copy the definition of the MER XML to the clipboard for the purpose of pasting into an external file.

If you are running the application in Microsoft Internet Explorer, a **Copy** button is provided on the **Export Monitor Event Request XML** dialog to copy the XML to the clipboard. If you are running the application in Mozilla Firefox, there is no **Copy** button because security features prevent browser access to the clipboard.

Deprecated Callout Interface

This appendix documents the callout interface that was deprecated in version 11.0.0.



The callout interface described in this appendix was deprecated in version 11.0.0 of the TIBCO iProcess Workspace (Browser). It is superseded by a simpler means of specifying default filters, sort, and column displays, which is described in [Callout Interface](#). New development should use the new callout interface.

This deprecated interface is still functional and can continue to be used. Note, however, that it is possible that the format of the serialized XML for columns and other XML representations of data could change in future releases of the product.

The sample file of these callout functions now reside in a file named `SampleCalloutHandlerDeprecated.js`.

Callout Interface

The callout interface provides methods that allow you to:

- Set default filters and sorts on the work item and case lists.
- Specify filters and sorts that can modify any user-defined filters and sorts.
- Specify which columns/fields the user can filter and sort on for work item and case lists.
- Set default columns to display on the procedure list, work queue list, case list, work item list, outstanding work item list on the case **Outstanding** tab, and the outstanding work item list on the **Process Jump** dialog.
- Specify which columns are available for a user to display from the **Column Selector** dialogs.

Methods in the callout interface can be used in combination with user access profile settings to control filter, sort, and column display. For example:

- You could use the callout interface methods to set a default filter on the case list, then use the access profiles to not allow the user to set a filter (i.e., do not give access to the case list **Filter** dialog).
- You could use the callout interface methods to display specific columns in the work item list by default, then use the user access profiles to not allow the user to change the columns (i.e., do not give access to the **Column Selector** on the work item list).

For information about user access profiles, see [User Access](#).

Since callout method calls are used to restrict access to data, any exceptions thrown will prevent the associated list from loading or close an already open list. An error message will be displayed to the user, logged to the Application Log and Application Monitor, and the list will be closed.

The callout methods are arranged in three functional groups:

- **Filter** - These methods control default filters and additional filters to apply to user-defined filters.
- **Sort** - These methods control default sorts and additional sorts to apply to user-defined sorts.
- **Column** - These methods control default column displays and which columns the user is allowed to select from the **Column Selector**.



Basic knowledge of XML, JavaScript, and TIBCO® General Interface is necessary to understand and work with the callout methods.

The following tables show the callout methods available:

Filter Methods (Case and Work Item Lists)

Method	Description
calloutInitialWorkItemFilter	Specifies the filter to be used when the work item list is

Method	Description
	opened. The filter that was saved when the list was last closed can be reused with or without modifications, or it can be replaced with a default filter. This filter appears on the Filter dialog after it is applied, and can be changed by the user if they have access to the Filter dialog.
calloutWorkItemFilter	Specifies the filter to apply to the work item list. This may be used to either modify the user-defined filter, to append additional criteria, or to override it. This is applied automatically when the user applies a filter by either clicking the Apply button on the work item list Filter dialog, or by applying a server-side find. Any user-defined filter will appear on the Filter dialog, but any modification applied to the filter with this method will not be visible to the user.
calloutWorkItemFilterColumns	Specifies which fields/columns can be used to filter work items, i.e., it controls which fields/columns appear in the Field drop-down list on the Filter dialog.
calloutInitialCaseFilter	Specifies the filter to be used when the case list is opened. The filter that was saved when the list was last closed can be reused with or without modifications, or it can be replaced with a default filter. This filter appears on the Filter dialog after it is applied, and can be changed by the user if they have access to the Filter dialog.
calloutCaseFilter	Specifies the filter to apply to the case list. This may be used to either modify the user-defined filter, to append additional criteria, or to override it. This is applied automatically when the user applies a filter by clicking the Apply button on the case list Filter dialog. Any user-defined filter will appear on the Filter dialog, but any modification applied to the filter with this method will not be visible to the user.

Method	Description
calloutCaseFilterColumns	Specifies which fields/columns can be used to filter cases, i.e., it controls which fields/columns appear in the Field drop-down list on the Filter dialog.

Sort Methods (Case and Work Item Lists)

Method	Description
calloutInitialWorkItemSort	Specifies the sort to be used when the work item list is opened. The sort that was saved when the list was last closed can be reused with or without modifications, or it can be replaced with a default sort. This sort appears on the Sort dialog after it is applied, and can be changed by the user if they have access to the Sort dialog.
calloutWorkItemSort	Specifies the sort to apply to the work item list. This may be used to either modify the user-defined sort, to append additional criteria, or to override it. This is applied automatically when the user applies a sort by clicking the Apply button on the work item list Sort dialog. Any user-defined sort will appear on the Sort dialog, but any modification applied to the sort with this method will not be visible to the user.
calloutWorkItemSortColumns	Specifies which fields/columns can be used to sort work items, i.e., it controls which fields/columns appear in the Available Fields list on the Sort dialog.
calloutInitialCaseSort	Specifies the sort to be used when the case list is opened. The sort that was saved when the list was last closed can be reused with or without modifications, or it can be replaced with a default sort. This sort appears on the Sort dialog after it is applied, and can be changed by the user if they have access to the Sort dialog.

Method	Description
calloutCaseSort	Specifies the sort to apply to the case list. This may be used to either modify the user-defined sort, to append additional criteria, or to override it. This is applied automatically when the user applies a sort by clicking the Apply button on the case list Sort dialog. Any user-defined sort will appear on the Sort dialog, but any modification applied to the sort with this method will not be visible to the user.
calloutCaseSortColumns	Specifies which fields/columns can be used to sort cases, i.e., it controls which fields/columns appear in the Available Fields list on the Sort dialog.

Column Methods (Procedure, Case, Work Queue, Work Item, Outstanding, Outstanding-Jump Lists)

Method	Description
calloutColumns	Specifies the default columns to display on the procedure list, case list, work queue list, work item list, outstanding work items list on the case Outstanding tab, and the outstanding work items lists on the Process Jump dialog.
calloutSelectColumns	Specifies which columns will be available to the user on the Column Selector dialog from the procedure list, case list, work queue list, work item list, outstanding work items list on the case Outstanding tab, and the outstanding work items lists on the Process Jump dialog. This controls which columns the user is able to display on each of the lists.

A custom class can implement one or more of the methods in the tables above. One or more custom classes may be used to handle these method calls.

If there is no implementation of these methods, there are no restrictions other than what might be applied through user access profiles.

Additional details about each callout method can be found in [Callout Method Signatures](#).



Also see the *Migration* section in the Release Notes for information about callout interface method considerations if you are upgrading your version of the iProcess Workspace (Browser).

Configuration

The TIBCO iProcess Workspace (Browser) comes with a sample callout handler that contains sample implementations of all of the callout methods. This sample callout handler is named 'SampleCalloutHandler.js' and is located in the *InstallationHomeDir*\iprocessclientbrowser\samples\Callouts directory, where *InstallationHomeDir* is the directory in which the installer places administrative files, such as the uninstaller, documentation, and sample code. This defaults to C:\tibco on Windows systems, and /opt/tibco on UNIX systems, but can be specified as a different directory when the iProcess Workspace (Browser) is installed.



Upon deprecation of this callout interface, the name of the sample callout handler file was renamed SampleCalloutHandlerDeprecated.js. For more information, see [Deprecated Callout Interface](#).

Perform the following steps to create a custom handler and configure your iProcess Workspace (Browser) to use the callout methods.

1. Copy the SampleCalloutHandler.js file into a directory you've created under the *ClientInstallDir*\JSXAPPS\ipc\ directory, where *ClientInstallDir* is the path to the directory in which the iProcess Workspace (Browser) is installed. For example, *ClientInstallDir*\JSXAPPS\ipc\Callouts.

You may also want to rename the SampleCalloutHandler.js file to identify the type of custom handling it performs. For example, 'ColumnsCalloutHandler.js'.

2. Modify the callout handler (e.g., ColumnsCalloutHandler.js) to fit your needs.

The original SampleCalloutHandler.js file that you copied contains sample implementations of each of the available callout methods.

Each callout method receives a data parameter that can be modified by the method and returned to the application. The following are example data parameters:

- filterExpression (string)

- `sortFields` (`Array<com.tibco.bpm.ipc.vSortField>`)
- `columns` (`jsx3.xml.Entity`)

Additional parameters provide information the methods can use to determine how the filters, sorts, and columns should be modified. For example:

- `username` (`string`)
- `eventNode` (`jsx3.xml.Entity`)
- `listType` (`string`)
- `availableFields` (`jsx3.xml.Entity`)

There is also a **componentName** parameter that specifies the specific instance of the component the method is affecting. This can be useful in WCC custom applications where you may be displaying multiple lists at one time, and would like to modify the filter, sort, or columns on only one of them.

The **jsx3.xml.Entity** object is a TIBCO® General Interface class that is a wrapper of the native browser XML node class. This class provides methods for querying, traversing, and creating XML entities (see the TIBCO® General Interface documentation for more information). The object is a Document Object Model (DOM) class that provides methods to add, find, modify, or delete XML values in an XML document. Use these methods to modify the incoming XML so that the desired filter, sort, or columns are displayed.

In each case, the method returns the same type of XML object that was passed in. This would probably be the same object in most cases, with some modification applied.

When customizing the callout handler, you must also register the callout method with the application **CalloutController** by adding the method to the `init` (constructor) method. It must be in the form:

```
app.getCalloutController().registerHandler
(target, arrayOfMethodNames)
```

where:

- *target* - (`Object`) The instance or object the method is called on.
- *arrayOfMethodNames* - (`Array<strings>`) Array of strings that are the names of the methods to register.

The following is an example of the `init` method in which the `calloutColumns` method is registered:

```
ipcClass.prototype.init = function(app) {
    this.app = app;
    this.controller = this.app.getCalloutController();
    this.controller.registerHandler(this,['calloutColumns']);
};
```

A reference to the application object is passed as the single parameter to the `init` (constructor) method.

Note that the application `getServer()` method can be used to get a reference to the **jsx3.app.Server** instance:

```
app.getServer()
```

3. Specify the callout handler custom class in the iProcess Workspace (Browser)'s configuration file, *ClientInstallDir\JSXAPPS\ipc\config.xml*.

The **<record jsxid="customCallout">** element specifies which classes will be loaded to handle custom callout methods. The **<Classes>** element can contain any number of **<Class>** elements whose **class** attribute is set to the fully qualified name of the custom class to load. The class is loaded after the user is authenticated at login. This gives the custom class access to the logged-in user's session to query the Action Processor for initialization data, if required.

The following is an example of the `customCallout` element identifying the `ColumnsCalloutHandler` custom class:

```
<record jsxid="customCallout" type="ipc">
    <Classes>
        <Class class="com.tibco.bpm.ipc.ColumnsCalloutHandler" />
    </Classes>
</record>
```

4. Add a mapping record to the `config.xml` file that points to the custom handler. This is added as a child element of the **<record jsxid="includes">** element. The following is an example class mapping element for the custom callout handler, `ColumnsCalloutHandler.js`.

```
<record jsxid="includes" type="array">
    ...
    <record jsxid="90" type="map">
        <record jsxid="id"
type="string">ColumnsCalloutHandler</record>
        <record jsxid="type" type="string">script</record>
```

```

        <record jsxid="owner" type="string">application</record>
        <record jsxid="onLoad" type="boolean">true</record>
        <record jsxid="required" type="boolean">true</record>
        <record jsxid="src"
type="string">JSXAPPS/ipc/Callouts/ColumnsCalloutHandler.js</recor
d>
        </record>
    </record>

```

- Optionally, modify the user access profiles that would be used in conjunction with the custom handling. For example, if your custom handler is setting the default columns on the work item list, you may want to deny access to the Column Selector on the work item list (see **SelectColumns** on [WorkQueue](#) [WorkItem](#) [SelectColumns](#)).



Note

Note that case is significant on some web servers, such as Tomcat. For example, if you are storing your custom callouts in the directory, *ClientInstallDir\JSXAPPS\ipc\Callouts* (i.e., with “Callouts” capitalized), the path specification to the custom callout handler in the config.xml file cannot be “JSXAPPS/ipc/callouts/ColumnsCalloutHandler.js” (i.e., with “callouts” not capitalized).

Callout Method Signatures

The following are the method signatures from the `SampleCalloutHandler.js` file (in JavaDoc format).

Note that the parameter data XML examples shown with the method signatures are representative samples — they may contain other attributes that are not shown.

calloutInitialWorkItemFilter

```

/**
 * @param filterExpression    (string) The filter string
value.
 * @param username           (string) The logged in user

```

```

name.
    * @param queueNode      (jsx3.xml.Entity) The queue node XML for
the
    *
workitem list.
    * @param availableFields (jsx3.xml.Entity) XML defining the
available
    *
fields that can be
filtered.
    * @param componentName  (string) Component instance
name
    *
    * @return                (string) Modified filter
string.
    */
ipcClass.prototype.calloutInitialWorkItemFilter = function(
filterExpression,
username,
queueNode,
availableFields
componentName)
{

```

calloutWorkItemFilter

```

/**
    * @param filterExpression  (string) The filter string
value.
    * @param username         (string) The logged in user
name.
    * @param queueNode      (jsx3.xml.Entity) The queue node XML for
the
    *
workitem list.
    * @param availableFields (jsx3.xml.Entity) XML defining the
available
    *
fields that can be
filtered.
    * @param componentName  (string) Component instance
name
    *
    * @return                (string) Modified filter
string.
    */
ipcClass.prototype.calloutWorkItemFilter = function

```



```

(filterExpression,
                                username,
queueNode,
                                availableFields
                                componentName)
{

```

The following is an example **filterExpression** parameter value used with **calloutWorkItemFilter**:

```
SW_PRONAME = "a*"
```

The sample above would show only work items whose procedure name starts with “a”. (For information about filter expression syntax, see the *TIBCO iProcess Server Objects (Java or .NET) Programmer’s Guide*.)

calloutWorkItemFilterColumns

```

/**
 * @param availableFields (jsx3.xml.Entity) XML defining the
available
 *                                fields that can be
filtered.
 * @param username          (string) The logged in user
name.
 * @param queueNode        (jsx3.xml.Entity) The queue node XML for
the
 *                                workitem list.
 * @param componentName    (string) Component instance
name
 *
 * @return                  (jsx3.xml.Entity) Modified XML defining
the
 *                                available fields
that can be
 *                                filtered.
 */
ipcClass.prototype.calloutWorkItemFilterColumns = function(
availableFields,
                                username,
                                queueNode
                                componentName)
{

```

The following are example **availableFields** and **queueNode** parameter values used with **calloutWorkItemFilterColumns**:

availableFields (jsx3.xml.Entity)

```
<data jsxid="jxroot">
  <record jsxid="OCCUPATION" jstext="Occupation" fieldType="swText"
    fieldLength="20" />
  <record jsxid="PM_ARRIVAL" jstext="Date and Time Arrived" fieldType="swDateTime"
    fieldLength="16" />
  <record jsxid="PM_ARRIVALDATE" jstext="Date Arrived" fieldType="swDate"
    fieldLength="10" />
  <record jsxid="PM_ARRIVALTIME" jstext="Time Arrived" fieldType="swTime"
    fieldLength="5" />
</data>
```

queueNode (jsx3.xml.Entity)

```
<record jsxid="IDA308B"
  Name="Admin"
  Description="System Administrator"
  MultiRecord="true"
  Type="ITAgentAdmin"
  InGroup="false"
  IDName="ID"
  FirstDeadline="2006-07-28 10:44:00"
  DeadlineCat="1"
  OpenedCat="13"
  OpenedCat="5"
  WorkItemCat="24"
  WorkParamName="WQ Parameter1"
  WorkParamName="WQ Parameter2"
  WorkParamName="WQ Parameter3"
  WorkParamName="WQ Parameter4" />
</record>
```

calloutInitialCaseFilter

```
/**
 * @param filterExpression      (string) The filter string
value.
 * @param username              (string) The logged in user
name.
 * @param procNode              (jsx3.xml.Entity) The procedure node XML
for
 *                               the case list.
 * @param availableFields      (jsx3.xml.Entity) XML defining the
available
 *                               fields that can be
sorted.
 * @param componentName        (string) Component instance
name
 *
 * @return                      (string) Modified filter
string.
 */
ipcClass.prototype.calloutInitialCaseFilter = function(
filterExpression,
username,
procNode,
```

```

availableFields
componentName)
{

```

calloutCaseFilter

```

/**
 * @param filterExpression      (string) The filter string
value.
 * @param username              (string) The logged in user
name.
 * @param procNode              (jsx3.xml.Entity) The procedure node XML
for
 *                               the case list.
 * @param availableFields (jsx3.xml.Entity) XML defining the
available
 *                               fields that can be
filtered.
 * @param componentName        (string) Component instance
name
 *
 * @return                      (string) Modified filter
string.
 */
ipcClass.prototype.calloutCaseFilter = function(filterExpression,
username,
procNode,
availableFields
componentName)
{

```

calloutCaseFilterColumns

```

    /**
     * @param availableFields (jsx3.xml.Entity) XML defining the
available
     *                               fields that can be
filtered.
     * @param username          (string) The logged in user
name.
     * @param procNode          (jsx3.xml.Entity) The proc node XML for
the
     *                               case list.
     * @param componentName      (string) Component instance
name
     *
     * @return                    (jsx3.xml.Entity) Modified XML defining
the      *                               available fields
that can be      *                               filtered.
     */
    ipcClass.prototype.calloutCaseFilterColumns = function
(availableFields,
                                username,
                                procNode
                                componentName)
{

```

calloutInitialWorkItemSort

```

    /**
     * @param sortFields          (Array) An array of
     *
     * com.tibco.bpm.ipc.vSortField
     *
     * instances.
     * @param username          (string) The logged in user
     name.
     * @param queueNode        (jsx3.xml.Entity) The queue node XML for
     the
     *
     * workitem list.
     * @param availableFields (jsx3.xml.Entity) XML defining the
     available
     *
     * fields that can be
     sorted.
     * @param componentName    (string) Component instance
     name
     *
     * @return                  (Array) Modified array of
     *
     * com.tibco.bpm.ipc.vSortField
     *
     * instances.
     */
    ipcClass.prototype.calloutInitialWorkItemSort = function
    (sortFields,
                                     username,
    queueNode,
    availableFields
                                     componentName)
    {

```

calloutWorkItemSort

```

/**
 * @param sortFields          (Array) An array of
 *
 * com.tibco.bpm.ipc.vSortField
 *
 * instances.
 * @param username            (string) The logged in user
name.
 * @param queueNode           (jsx3.xml.Entity) The queue node XML for
the
 *
 * workitem list.
 * @param availableFields (jsx3.xml.Entity) XML defining the
available
 *
 * fields that can be
sorted.
 * @param componentName       (string) Component instance
name
 *
 * @return                    (Array) Modified array of
 *
 * com.tibco.bpm.ipc.vSortField
 *
 * instances.
 */
ipcClass.prototype.calloutWorkItemSort = function(sortFields,
                                                    username,
queueNode,
availableFields
                                                    componentName)
{

```

The following describes the **sortFields** parameter used with **calloutWorkItemSort**:

Each **vSortField** has three properties with accessors as shown:

- fieldName getFieldName()
- ascending getAscending()
- sortAsType getSortAsType()

For example:

```

sortFields[0] :
fieldName : SW_CASEDESC
ascending : true
sortAsType: swTextSort

```

```

sortFields[1] :
fieldName : SW_CASENUM
ascending : true
sortAsType: swTextSort

```

Work items will be sorted in the order in which elements are passed in the **vSortField** array.

New **vSortField** values are created by passing the three properties in the constructor:

```

var newSortFields = new Array();
newSortFields.push(new com.tibco.bpm.ipc.vSortField('SW_CASEDESC',
                                                    true,
                                                    'swTextSort'));
newSortFields.push(new com.tibco.bpm.ipc.vSortField('SW_CASENUM',
                                                    true,
                                                    'swTextSort'));

```

calloutWorkItemSortColumns

```

/**
 * @param availableFields (jsx3.xml.Entity) XML defining the
available
 *
 *                               fields that can be
sorted.
 * @param username          (string) The logged in user
name.
 * @param queueNode        (jsx3.xml.Entity) The queue node XML for
the
 *                               workitem list.
 * @param componentName    (string) Component instance
name
 *
 * @return                  (jsx3.xml.Entity) Modified XML defining
the
 *                               available fields
that can be
 *                               filtered.
 */
ipcClass.prototype.calloutWorkItemSortColumns = function(

availableFields,

                                username,
                                queueNode
                                componentName)
{

```

The following is an example **availableFields** parameter value used with **calloutWorkItemSortColumns**:

```
<data jsid="jsroot">
  <record jsid="IN_ARRIVAL" jstext="Date and Time Arrived"
    sorttype="dateTimeSort" />
  <record jsid="IN_CASEDESC" jstext="Case Description"
    sorttype="textSort" />
  <record jsid="IN_CASENUM" jstext="Case Number"
    sorttype="numericSort" />
  ...
</data>
```

calloutInitialCaseSort

```
/**
 * @param sortFields          (Array) An array of
 *
 * com.tibco.bpm.ipc.vSortField
 *
 * instances.
 * @param username            (string) The logged in user
 * name.
 * @param procNode            (jsx3.xml.Entity) The procedure node XML
 * for
 *
 * the case list.
 * @param availableFields    (jsx3.xml.Entity) XML defining the
 * available
 *
 * fields that can be
 * sorted.
 * @param componentName       (string) Component instance
 * name
 *
 *
 * @return                    (Array) Modified array of
 *
 * com.tibco.bpm.ipc.vSortField
 *
 * instances.
 */
ipcClass.prototype.calloutInitialCaseSort = function(sortFields,
username,
procNode,
availableFields
componentName)
{
```

calloutCaseSort

```
/**
 * @param sortFields          (Array) An array of
 *
 *
```



```

com.tibco.bpm.ipc.vSortField
    *
    * @param username (string) The logged in user
name.
    * @param procNode (jsx3.xml.Entity) The procedure node XML
for
    *
    * the case list.
    * @param availableFields (jsx3.xml.Entity) XML defining the
available
    *
    * fields that can be
sorted.
    * @param componentName (string) Component instance
name
    *
    * @return (Array) Modified array of
    *
com.tibco.bpm.ipc.vSortField
    *
    */
    ipcClass.prototype.calloutCaseSort = function(sortFields,
                                                username,
procNode,
availableFields
                                                componentName)
{

```

calloutCaseSortColumns

```

    /**
    * @param availableFields (jsx3.xml.Entity) XML defining the
available
    *
    * fields that can be
sorted.
    * @param username (string) The logged in user
name.
    * @param procNode (jsx3.xml.Entity) The proc node XML for
the
    *
    * case list.
    * @param componentName (string) Component instance
name
    *
    * @return (jsx3.xml.Entity) Modified XML defining
the
    *
    * available fields
that can be
    *
    * filtered.
    */

```

```

        ipcClass.prototype.calloutCaseSortColumns = function
        (availableFields,
                                                    username,
                                                    procNode
                                                    componentName)
    {

```

calloutColumns

```

        /**
        * @param columns      (jsx3.xml.Entity) The serialized columns
for
        *
        *                               the list.
        * @param username      (string) The logged in user
name.
        * @param eventNode      (jsx3.xml.Entity) The procedure (Cases
list),
        *                               workQ (WorkItems
list), or
        *                               caseTag data
        *                               (Outstanding)
        *                               node XML. Null for
Proc and
        *                               WorkQ list types:
        * @param availableFields (jsx3.xml.Entity) XML defining the
available
        *                               fields for column
selection.
        * @param listType (string) The list type, one of:
        *                               com.tibco.bpm.ipc.ListContainer.PROC
        *                               com.tibco.bpm.ipc.ListContainer.CASE
        *                               com.tibco.bpm.ipc.ListContainer.WORKQ
        *                               com.tibco.bpm.ipc.ListContainer.WORKITEM
        *                               com.tibco.bpm.ipc.ListContainer.OUTSTANDING
        *                               com.tibco.bpm.ipc.ListContainer.OUTSTANDING + 'Jump'
        * @param componentName      (string) Component instance
name
        *
        * @return      (jsx3.xml.Entity) Modified serialized
columns
        *                               for the list.
        */

```

```

ipcClass.prototype.calloutColumns = function(columns,
                                             username,
                                             eventNode,
                                             availableFields,
                                             listType
                                             componentName)
{

```

The following describes the **eventNode** parameter used with **calloutColumns**:

The value of **eventNode** depends on the type of list as shown below:

- Proc list : null
- Case list:

```

<!--case-->
Name="ALLOCATE"
Description="Allocate Resources"
RoutingNode="12tagtest"
Version="1.2"
Tap="12tagtest|ALLOCATE|012"
FromNumber="54"
StartStepName="STEP1"
Status="NoRelease"
CaseDescOpt="NoRequiredDesc"
IsAutoExec="false"
IsIgnoreBlank="false"
IsAutoExec="false"
IsSubProcess="false"
IsFromName="false"
IsWorkflow="true"
IsWorkflow="false"
Owner="swabala"
Description="DefinitionName"
Permission="Start / History"
CaseCount="44"
ActiveCount="19"
ClosedCount="1"
</!--case-->

```

- WorkQ list: null
- WorkItem list:

```

<!--work-->
Name="12TestGroup"
Description="12TestGroup"
RoutingNode="12tagtest"
Tap="12tagtest|12TestGroup|8"
InfoGroup="Type"
IsReleased="true"
DeadLineCnt="0"
Unprocessed="1"
UrgentCnt="0"
WorkItemCnt="1"
WorkParamName="WQ Parameter1"
WorkParamName="WQ Parameter2"
WorkParamName="WQ Parameter3"
WorkParamName="WQ Parameter4"
</!--work-->

```

- Outstanding or Outstanding Jump:

```

<!--out-->
CaseTap="12tagtest|ALLOCATE|012|2453"
NodeName="12tagtest"
FromName="ALLOCATE"
MajorVersion="0"
MinorVersion="0"
CaseNumber="2453"
</!--out-->

```

calloutSelectColumns

```

/**
 * @param availableFields (jsx3.xml.Entity) XML defining the
fields
 *
 * available for column
 *
 * selection.
 * @param username (string) The logged in user
name.
 * @param eventNode (jsx3.xml.Entity) The procedure (Cases
list),
 *
 * workQ (WorkItems
list), or
 *
 * caseTag data
 *
 * (Outstanding)
 *
 * node XML. Null for
Proc and
 *
 * WorkQ list types:
 * @param columns (jsx3.xml.Entity) The serialized columns
for
 *
 * the list.
 * @param listType (string) The list type, one of:
 *
 * com.tibco.bpm.ipc.ListContainer.PROC
 *
 * com.tibco.bpm.ipc.ListContainer.CASE
 *
 * com.tibco.bpm.ipc.ListContainer.WORKQ
 *
 * com.tibco.bpm.ipc.ListContainer.WORKITEM
 *
 * com.tibco.bpm.ipc.ListContainer.OUTSTANDING
 *
 * com.tibco.bpm.ipc.ListContainer.OUTSTANDING + 'Jump'
 * @param componentName (string) Component instance
name
 *
 * @return (jsx3.xml.Entity) Modified XML defining
the
 *
 * fields available for
column
 *
 * selection.
 */
ipcClass.prototype.calloutSelectColumns = function
(availableFields,
                                username,
eventNode,
                                columns,
                                listType

```

```

componentName)
{

```

The following are example **availableFields** and **columns** parameter values used with **calloutSelectColumns**:

availableFields (jsx3.xml.Entity)

```

<data jsxid="jsxroot">
  <record jsxid="IsStatusImage" />
  <record jsxid="CaseNumber" fieldname="BU_CASENUM" fieldtype="collname" />
  <record jsxid="CaseReference" fieldname="BU_CASEREF" fieldtype="text" />
  ...
</data>

```

columns (jsx3.xml.Entity)

The columns value contains the serialized columns for the list. The following sample shows how this can be obtained from a **jsx3.gui.List**:

```

var objProperties = new Object();
objProperties['children'] = true;
var serializedXml = jsxList.toXML(objProperties);
The following is a sample of the serialized columns:

```

```

<?xml version="1.0" encoding="utf-8" ?>

<serialization xmlns="urn:tibco.com/v3.0" jsxversion="3.5">
  <name><CDATA[IsStatusImage]</name>
  <icon><CDATA[IsStatusImage]</icon>
  <description><CDATA[IsStatusImage]</description>
  <onBeforeDeserialize><CDATA[IsStatusImage]</onBeforeDeserialize>
  <onAfterDeserialize><CDATA[IsStatusImage]</onAfterDeserialize>
  <object type="jsx3.gui.Matrix.Column">
    <variants jsxwidth="24" />
    <strings jsxname="collStatusImage" jsxpath="IsStatusImage" ... />
    <dynamics jsxhg="tpcCollHeader BG" jsxborder="0" jsxoutlet="0" />
  </object>
  <object type="jsx3.gui.Matrix.Column">

```

TIBCO Documentation and Support Services

For information about this product, you can read the documentation, contact TIBCO Support, and join TIBCO Community.

How to Access TIBCO Documentation

Documentation for TIBCO products is available on the [TIBCO Product Documentation](#) website, mainly in HTML and PDF formats.

The [TIBCO Product Documentation](#) website is updated frequently and is more current than any other documentation included with the product.

Product-Specific Documentation

Documentation for TIBCO iProcess® Workspace (Browser) is available on the [TIBCO iProcess® Workspace \(Browser\) Product Documentation](#) page.

The following documents for this product can be found in the TIBCO Documentation site:

- *TIBCO iProcess® Workspace (Browser) Release Notes*
- *TIBCO iProcess® Workspace (Browser) Installation*
- *TIBCO iProcess® Workspace (Browser) User Guide*
- *TIBCO iProcess® Workspace (Browser) Action Processor Reference*
- *TIBCO iProcess® Workspace (Browser) Components Concepts*
- *TIBCO iProcess® Workspace (Browser) Components Reference*
- *TIBCO iProcess® Workspace (Browser) Configuration and Customization Guide*
- *TIBCO iProcess® Workspace (Browser) Integrating TIBCO Forms 2.1.x with GI Applications*

How to Contact TIBCO Support

Get an overview of [TIBCO Support](#). You can contact TIBCO Support in the following ways:

- For accessing the Support Knowledge Base and getting personalized content about

products you are interested in, visit the [TIBCO Support](#) website.

- For creating a Support case, you must have a valid maintenance or support contract with TIBCO. You also need a user name and password to log in to [TIBCO Support](#) website. If you do not have a user name, you can request one by clicking **Register** on the website.

How to Join TIBCO Community

TIBCO Community is the official channel for TIBCO customers, partners, and employee subject matter experts to share and access their collective experience. TIBCO Community offers access to Q&A forums, product wikis, and best practices. It also offers access to extensions, adapters, solution accelerators, and tools that extend and enable customers to gain full value from TIBCO products. In addition, users can submit and vote on feature requests from within the [TIBCO Ideas Portal](#). For a free registration, go to [TIBCO Community](#).

Legal and Third-Party Notices

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNDLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE “LICENSE” FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO, the TIBCO logo, the TIBCO O logo, and iProcess are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Java and all Java based trademarks and logos are trademarks or registered trademarks of Oracle Corporation and/or its affiliates.

This document includes fonts that are licensed under the SIL Open Font License, Version 1.1, which is available at: <https://scripts.sil.org/OFL>

Copyright (c) Paul D. Hunt, with Reserved Font Name Source Sans Pro and Source Code Pro.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

This software may be available on multiple operating systems. However, not all operating system platforms for a specific software version are released at the same time. See the readme file for the availability of this software version on a specific operating system platform.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

This and other products of TIBCO Software Inc. may be covered by registered patents. Please refer to TIBCO's Virtual Patent Marking document (<https://www.tibco.com/patents>) for details.

Copyright © 2005-2022. TIBCO Software Inc. All Rights Reserved.